FACULDADE DE CIÊNCIAS SOCIAIS E HUMANAS
UNIVERSIDADE NOVA DE LISBOA

Desenho de interface para o desenvolvimento do pensamento computacional no Ensino Básico: análise do Scratch

Sarita Bastos Costa

Dissertação de Mestrado em Novos Media e Práticas Web

Nota: Sarita Bastos Costa, Desenho de interface para o desenvolvimento do pensamento computacional no Ensino Básico: análise do Scratch, 2016

Abril, 2016

Dissertação apresentada para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Novos Media e Práticas Web, realizada sob a orientação científica da Profª Doutora Graça Rocha Simões, docente do departamento de Ciências da Comunicação da Faculdade de Ciências Sociais e Humanas da Universidade Nova de Lisboa (FCSH/NOVA).

“Anything is easy if you can assimilate it to your collection of models. If you can't, anything can be painfully difficult.”

Seymour Papert

Aos meus pais, Antonio (in memoriam) e Maria de Fátima (in memoriam)

AGRADECIMENTOS

À minha orientadora, professora doutora Graça Rocha Simões, pelo incentivo e indicações preciosas para a dissertação.

À minha família, especialmente a minha irmã Rita de Cássia, pela confiança e reconhecimento da importância da pesquisa para a minha evolução acadêmica e profissional.

Ao meu namorado, João Mendonça, que compreendeu minha necessidade de isolamento para o estudo e ofereceu o suporte necessário desde o início do mestrado.

As palavras finais vão para minha mãe. Sem o seu amor, incentivo e confiança, eu não teria a determinação necessária para concluir o mestrado. A senhora faleceu no meio deste processo, mas deixou semeada a força para que cada um de nós, filhos, pudesse seguir seus próprios caminhos. Obrigada.

DISSERTAÇÃO

Desenho de interface para o desenvolvimento do pensamento computacional no Ensino Básico: análise do Scratch

SARITA BASTOS COSTA

RESUMO

O pensamento computacional é a habilidade de aplicar métodos e técnicas das Ciências da Computação para resolver problemas em outras ciências, profissões e vida cotidiana. Nos últimos anos, acumulam-se evidências que reforçam os argumentos económicos, sociais e educativos para o desenvolvimento do pensamento computacional no Ensino Básico. Esse desafio foi impulsionado, em boa parte, pelo artigo *Computational Thinking*, escrito por Jeannete Wing, em 2006, em que a autora aponta as razões pelas quais ter a capacidade de pensar computacionalmente será tão importante quanto ler, escrever e fazer operações aritméticas. O crescente consenso sobre a importância do tema tem aproximado entidades profissionais, educadores, governos e organizações privadas para o desenvolvimento de metodologias e ferramentas adequadas para a promoção do pensamento computacional em ambientes formais e informais de aprendizagem. Um traço comum às diferentes iniciativas é o uso de recursos tecnológicos, como linguagens de programação, jogos digitais e kits robóticos. Diversos estudos têm sido conduzidos para avaliar as potencialidades pedagógicas dessas ferramentas, mas poucos indicam os melhores caminhos para desenvolvê-las. Nesta dissertação, indicamos os princípios que devem orientar o desenho de interfaces de softwares educativos para o desenvolvimento do pensamento computacional no Ensino Básico. Para tanto, reunimos relevantes estudos provenientes do campo da Interação Homem-Computador que fornecem orientações para o desenho, implementação e avaliação de interfaces de aprendizagem apropriadas à idade e ao desenvolvimento cognitivo da audiência visada. Demonstramos, por meio da análise do Scratch, o resultado concreto de decisões de *design* guiadas por esses princípios.

PALAVRAS-CHAVE: Pensamento computacional, Desenho de interface, Aprendizagem, Interação Criança-Computador, Desenho Centrado no Aprendiz.

ABSTRACT

Computational thinking is the skill to apply methods and techniques from Computer Science in order to solve problems in other sciences, professions and daily life. In the last years, evidences reinforce economic, social and educational arguments for the development of computational thinking in elementary school. A challenge that was sparked by the article *Computational Thinking*, written by Jeannete Wing, in 2006, in which the author points the reasons why having the ability to think computationally is as important as being capable of read, write or solve aritmetic operations. A wide agreement over the importance of this assumption has lead professional entities, educators, governments and private organizations to elaborate methodologies and tools to promote computational thinking in formal and informal learning environments. It is common to the different initiatives the use of technological resources, like programming languages, digital games and robotic kits. Several studies have been assembled in order to evaluate the potential pedagogic benefits, but only a few of those studies indicates how to design it. This dissertation indicates the design principles that should guide the designing of educational software to promote computational thinking in elementary school. We considered relevant studies from the field of Human Computer Interaction that provided guidelines to the design, implementation and evaluation of leaning interfaces appropriate to the age and to the cognitive development of the target audience. We demonstrate, by analysing Scratch, the efective results of design decisions guided by those principles.

KEYWORDS: Computational thinking, Design, User interface, Learning, Child-Computer Interaction, Learner-centered design.

ÍNDICE

Introdução

Há 10 anos, Jeannette Wing escreveu um manifesto em defesa do ensino do pensamento computacional para todas as pessoas e não apenas para os cientistas da computação. Seu influente artigo *Computational Thinking* promove a ideia de que todos podem aproveitar as vantagens dos métodos usados em Ciências da Computação para a resolução de problemas, desenvolvimento de sistemas e compreensão do comportamento humano. Para Wing, a habilidade de pensar logicamente, reconhecer padrões, fazer abstrações, decompor um problema e apresentar soluções de forma algorítmica será, para o cidadão do século XXI, tão importante quanto ler, escrever e fazer operações aritméticas básicas.

O argumento em defesa do desenvolvimento do pensamento computacional remonta às ideias de Seymour Papert. Em 1967, Papert desenvolveu, junto com os seus colegas do MIT, a linguagem de programação LOGO, desenhada especialmente para crianças. Em sua visão, a prática da programação estimularia as crianças a desenvolver o pensamento analítico, uma das características do pensamento computacional.

Apesar do pioneirismo de Papert, o artigo de Wing ajudou a popularizar o termo. Desde sua publicação, acumulam-se evidências que reforçam os argumentos econômicos, sociais e educativos para o desenvolvimento do pensamento computacional desde a infância. O consenso em torno do assunto tem aproximado educadores, agentes políticos e empresas de tecnologia, como Microsoft e Google, para o desenvolvimento de metodologias e ferramentas adequadas para a promoção do pensamento computacional em ambientes formais e informais de aprendizagem.

No contexto europeu, a tendência é confirmada pelo relatório *Computing our Future: Computer Programming and Coding* (2015) lançado pela rede europeia dos Ministérios da Educação (*European Schoolnet*). O relatório apresenta uma visão geral da integração do ensino de programação e pensamento computacional em sistemas educativos de 20 países europeus e Israel. O Reino Unido, por exemplo, inseriu o pensamento computacional nas competências digitais que devem ser desenvolvidas desde os primeiros anos de escolaridade. Em Portugal, destaca-se o projeto-piloto "Iniciação à Programação no 1.º Ciclo do Ensino Básico", promovido pela Direção-Geral da Educação.

Um traço comum às diferentes iniciativas é o uso de recursos tecnológicos para o desenvolvimento das diferentes habilidades relacionadas ao pensamento computacional, seja por meio de linguagens de programação visual direcionadas para crianças, de jogos ou por meio de *kits* robóticos. Diversos estudos têm sido conduzidos para avaliar as potencialidades pedagógicas dessas ferramentas, mas poucos indicam os melhores caminhos para desenvolvê-las. Observando este cenário com as lentes do *designer* de sistemas interativos, surge a questão norteadora desta dissertação: como desenhar interfaces de *softwares* educativos para o desenvolvimento do pensamento computacional no Ensino Básico?

Para respondê-la, é necessário recorrer, inicialmente, ao recurso da decomposição da questão inicial. A partir desse procedimento, identificamos quatro aspectos fundamentais que implicam outras questões de investigação:

1. Desenho de interfaces: Qual as melhores recomendações para o desenho de interfaces de sistemas interativos?

2. *Softwares* educativos: Como desenhar interfaces de aprendizagem?

3. Pensamento computacional: Que características do pensamento computacional podem ser desenvolvidas por meio da utilização de *softwares* educativos? Quais as melhores referências?

4. Ensino Básico: Quais as diretrizes curriculares que orientam o ensino do pensamento computacional no Ensino Básico? Como desenhar interfaces para crianças em ambientes formais de aprendizagem?

Estes quatros aspectos ajudaram a compor os núcleos temáticos da dissertação: pensamento computacional, aprendizagem e desenho de interfaces. O estudo sobre cada tema possibilitou a criação de um quadro de referência teórico a partir do qual analisamos os componentes da interface do Scratch, uma ferramenta que permite o desenvolvimento de diversas dimensões do pensamento computacional.

A escolha do Scratch foi motivada por sua popularidade entre os estudantes e pela boa avaliação entre os educadores. Desenvolvida por pesquisadores do Lifelong Kindergarten Group, do MIT Media Lab, a plataforma registra mais de 13 milhões de projetos e 10 milhões de usuários, a maioria na faixa etária de 10 a 12 anos.

Scratch cria as condições necessárias para a construção de artefatos por meio de métodos e técnicas usados em Ciências da Computação. Acreditamos que a análise da plataforma gera boas referências para o desenho de interfaces de desenvolvimento do pensamento computacional no Ensino Básico.

Motivação para a pesquisa

Alguns acontecimentos foram fundamentais para a escolha deste tema de pesquisa. O primeiro foi a participação no curso *Learning Creative Learning* (http://learn.media.mit.edu/), promovido no início de 2013 por pesquisadores do Lifelong Kindergarten, grupo de pesquisa do MIT Media Lab. Foi a partir do curso que conheci o trabalho visionário de Seymour Papert, o sistema Logo e a plataforma Scratch.

A motivação ganhou impulso no seminário Standards de Usabilidade e Acessibilidade do mestrado Novos Media e Práticas Web da Universidade Nova de Lisboa. Durante os estudos sobre os diferentes paradigmas da Interação Homem-Computador, chamou a atenção o fato de o sistema Logo também ter contribuído para a evolução das interfaces dos sistemas interativos. O interesse pelo tema levou a uma observação mais atenta sobre o desenho de interfaces para crianças e para as iniciativas de ensino de competências digitais para este público.

A pesquisa também foi motivada por minha trajetória profissional. Formada em jornalismo em 2007, logo comecei a atuar na análise de mídias sociais. Desenvolver habilidades do pensamento computacional é imprescindível para a área, assim como será para a maioria das profissões, como indicou Jeanette Wing.

Em um mundo marcado pela computação ubíqua e pela constante evolução tecnológica, é necessário que pesquisadores de diversas áreas, sobretudo das Ciências da Computação, Interação Homem-Computador, Pedagogia e Comunicação contribuam com pesquisas e ideias que estimulem o desenvolvimento do pensamento computacional desde a infância.

Estrutura da dissertação

Esta dissertação está dividida em seis seções: Introdução; Pensamento Computacional; Pensar sobre aprendizagem; Recomendações de *design* para *softwares* educativos; Scratch; e Conclusões.

N a *Introdução*, realizamos a contextualização do tema e apresentamos as principais questões de investigação, assim como os núcleos temáticos do trabalho. A seguir, explicamos as motivações para a pesquisa e a organização da dissertação.

No capítulo 1, intitulado *Pensamento Computacional*, faz-se uma revisão das principais definições de pensamento computacional, suas características e a presença em diversas áreas científicas e profissionais. Em seguida são abordadas as iniciativas para o ensino e as diretrizes curriculares que orientam a incorporação do pensamento computacional no rol das competências digitais desenvolvidas no Ensino Básico.

No capítulo 2, intitulado *Pensar sobre aprendizagem*, apresentamos os fundamentos das principais teorias da aprendizagem. Parte-se do pressuposto de que, para desenhar sistemas interativos no contexto educacional, os *designers* precisam compreender como as pessoas aprendem. Após um percurso pelas teorias, o capítulo prossegue com os principais paradigmas que orientaram o desenvolvimento de tecnologias educativas.

No capítulo 3, intitulado *Recomendações de design para softwares educativos*, apresentamos conceitos e recomendações para o desenho de interface de aprendizagem elaborados em pesquisas sobre Desenho Centrado no Aprendiz e sobre Interação Criança-Computador. Também são abordadas as boas práticas do desenho de interfaces baseadas em ferramentas que promovem o pensamento computacional.

No capítulo 4, intitulado *Scratch*, analisamos os princípios que guiaram as decisões sobre desenho da interface deste programa específico. Iniciamos por uma descrição das características gerais da ferramenta e do contexto em que foi desenvolvida. Em seguida, apresentamos a teoria da aprendizagem que influenciou os seus desenvolvedores. Então identificamos, a partir da análise dos componentes da interface, a estratégia usada pelo Scratch para ajudar os estudantes a desenvolver o pensamento computacional.

Por último, na *Conclusão*, são apresentadas as principais conclusões da investigação efectuada e recomendações para o desenho de interfaces para a introdução do pensamento computacional.

Capítulo 1: Pensamento computacional

1.1 O que é pensamento computacional?

Em março de 2006, Jeannette Wing, diretora do Center for Computational Thinking da Carnegie Mellon University, publicou o artigo *Computational Thinking* para defender a ideia de que o pensamento computacional deveria ser incluído na formação de todos os meninos e meninas por ser uma aptidão fundamental para cidadãos do século XXI, tão importante quanto ler, escrever e fazer operações aritméticas. A autora afirma que pensamento computacional é uma abordagem para a resolução de problemas, concepção de sistemas e compreensão do comportamento humano que tem como base conceitos fundamentais das Ciências da Computação (2006, p.33).

O termo pensamento computacional aparece em outras referências acadêmicas das Ciências da Computação em décadas anteriores. Seymour Papert, um dos pioneiros da inteligência artificial, faz referência ao termo em seu famoso livro *Mindstorms: Children, Computers, and Powerful Ideas* (1980), em que apresenta sua visão sobre aprendizagem e computação. No entanto, o artigo de Jeannette Wing é considerado o propulsor da atual mobilização de educadores, entidades profissionais, empresas e pesquisadores das Ciências da Computação interessados em por em prática o desafio de ensinar a pensar computacionalmente.

Em artigos mais recentes (2008; 2010; 2014), Wing define pensamento computacional como um processo de pensamento envolvido na formulação de problemas e expressão de suas soluções de tal forma que essas soluções possam ser eficazmente executadas por um agente de processamento de informação. A autora ressalta que o agente de processamento da informação pode ser tanto uma máquina quanto um ser humano:

> Informalmente, o pensamento computacional descreve a atividade mental na formulação de um problema de modo que possa ser admitida uma solução computacional. A solução pode ser levada a cabo por uma máquina ou um ser humano. Este último ponto é importante. Em primeiro lugar, os seres humanos computam. Em segundo lugar, as pessoas podem aprender pensamento computacional sem recurso a uma máquina. Além disso, o pensamento computacional não é apenas sobre a resolução de problemas, mas também sobre a formulação do problema. (Wing, 2014, para. 2, tradução livre)

Aho (2012) simplifica esta definição conceituando pensamento computacional como os "processos do pensamento envolvidos na formulação de problemas, de modo

que suas soluções possam ser representadas como passos e algoritmos computacionais" (p. 2). O que é crucial neste processo, segundo o autor, é encontrar ou desenvolver modelos de computação apropriados para formular os problemas e suas soluções.

A organização britânica The Royal Society também apresenta uma definição concisa. Para a organização, o pensamento computacional é o "processo de reconhecer os aspectos da computação no mundo que nos rodeia, a aplicação de ferramentas e técnicas de Ciências da Computação para entender os sistemas e processos naturais e artificiais" (Royal Academy of Engineering, 2012, p. 29).

É importante distinguir pensamento computacional de outros conceitos e práticas associados às competências digitais. Philips (2008) faz uma lista de aspectos que não podem ser confundidos com pensamento computacional:

- Não é apenas mais detalhes técnicos para a utilização de software;
- Não é pensar como um computador;
- Não é programação (necessariamente);
- Nem sempre requer um computador;
- Não é mais uma "coisa" para acrescentar ao currículo.

Para os propósitos desta dissertação, apoiamo-nos na definição operacional adotada pela International Society for Technology in Education (ISTE) e pela Computer Science Teachers Association (CSTA) que oferece um quando de referência útil para a incorporação do pensamento computacional em programas curriculares do ensino básico. Segundo esta definição, o pensamento computacional é um processo de resolução de problemas que consiste (mas não se limita) nas seguintes características:

1. Formular problemas de forma que permita usar um computador e outras ferramentas para solucioná-los;
2. Organizar e analisar dados de forma lógica;
3. Representar dados através de abstrações, como modelos e simulações;
4. Automatizar soluções através de pensamento algorítmico (uma série de passos ordenados);

5. Identificar, analisar e implementar soluções possíveis com o objetivo de alcançar a mais eficiente combinação de passos e recursos;

6. Generalizar e transferir esse processo de resolução de problemas a uma grande variedade de um mesmo tipo de problema. (CSTA, 2011).

ISTE e CSTA (2011) também apresentam as atitudes e predisposições necessárias para o desenvolvimento do conjunto de habilidades inerentes ao pensamento computacional:

1. A confiança em lidar com a complexidade;

2. Persistência em trabalhar com problemas difíceis;

3. Tolerância à ambiguidade;

4. A capacidade de lidar com problemas abertos e finitos;

5. A capacidade de comunicar e trabalhar com outros para atingir um objetivo comum ou solução. (CSTA, 2011).

1.2 Características do pensamento computacional

É possível observar traços comuns aos diferentes conceitos de pensamento computacional. Selby e Woollard (2014) identificam, a partir de uma revisão de literatura, os termos que prevalecem nas produções acadêmicas. Para os autores, parece haver um consenso de que a definição de pensamento computacional deve incluir a ideia de um processo de pensamento, o conceito de abstração e o conceito de decomposição. Além desses três termos, que são usados com frequência na literatura científica, é possível identificar outros três: desenho de algoritmo, generalização e avaliação. Vejamos as características mais importantes de cada um deles:

a) Processos de pensamento

Pensar computacionalmente envolve processos cognitivos usados em Ciências da Computação para a formulação e resolução de problemas. Esta característica é apontada nas definições de pensamento computacional supracitadas (Wing, 2011; Aho, 2012; Royal Academy of Engineering, 2012).

b) Abstração

Para Wing (2011), o mais importante processo cognitivo necessário para o pensamento computacional é a abstração. O uso de abstração é necessário para capturar e ressaltar propriedades essenciais comuns a um conjunto de objetos ou dados. A habilidade de destacar aspectos essenciais por meio da abstração é fundamental para criar um modelo representativo, como um mapa das linhas de metrô, desenhar algoritmo, gerar gráficos e outras formas de visualização de dados. Portanto, a habilidade de abstrair ajuda a lidar com problemas complexos por meio da redução de detalhes desnecessários (Catlin & Woollard, 2014).

c) Decomposição

Decomposição é a habilidade de dividir dados, processos ou problemas em partes menores e mais fáceis de serem resolvidas. É uma forma de pensar sobre os artefatos a partir de suas partes componentes. Essa habilidade cognitiva ajuda a resolver problemas complexos, a lidar com situações novas e a projetar sistemas (Google, n.d.; Selby & Woollard, 2014).

d) Algoritmo

O termo algoritmo é interpretado como uma sequência de instruções necessárias para a realização de uma tarefa. Desenvolver o pensamento algorítmico ajuda a identificar e a elaborar uma sequência de eventos para alcançar uma solução ou entender um fenômeno (Selby & Woollard, 2014).

e) Generalização

Generalização é uma característica descrita como a capacidade de expressar a solução de um problema em termos genéricos, que pode ser aplicada em outras situações que compartilham características do problema original (Selby & Woollard, 2014). Também está associada à capacidade de reconhecer padrões, similaridades e conexões.

f) Avaliação

Pensar computacionalmente também requer uma atitude de constante avaliação das decisões tomadas, dos resultados e do estado do sistema. Selby & Woollard (2014) preferem o termo avaliação à análise.

1.3 Pensamento computacional em diferentes áreas científicas e profissionais

Há muitos exemplos da aplicação do pensamento computacional em diversas disciplinas, profissões e atividades do cotidiano. O documento *Computational Thinking Leadership Toolkit* (CSTA, 2011) apresenta algumas das situações mais comuns:

a) Quando um escritor pesquisa um assunto na internet e registra suas notas em uma base de dados pessoal em seu *notebook*;

b) Quando um empresário usa dados censitários governamentais disponíveis na internet para estimar o potencial de mercado de um novo produto;

c) Quando um cientista da computação desenvolve modelos e simulações para representar sistemas biológicos complexos.

Apresentamos, a seguir, outros exemplos de como a aplicação de métodos das Ciências da Computação está revolucionando diferentes áreas do conhecimento.

- Biologia

O Projeto Genoma Humano, iniciado em 1990 pelo Departamento de Energia dos Estados Unidos, explicita a importância do uso do pensamento computacional no campo da Biologia (Wing, 2011). Para alcançar o objetivo de sequenciar todo o DNA do genoma humano foi necessário o uso de algoritmos e modelos computacionais. A união de esforços interdisciplinar ajudou a criar as bases para as disciplinas Bioinformática e Biologia Computacional.

- Jornalismo

O pensamento computacional também é necessário para que os jornalistas possam atender aos novos desafios da profissão. É inegável a dependência do jornalismo das ferramentas computacionais antes mesmo da recente revolução digital. A busca por informações em bases de dados governamentais ou científicos não é uma prática recente para os jornalistas. A "Reportagem Assistida por Computador" (*Computer-Assisted Reporting*), por exemplo, foi uma das primeiras abordagens sobre o uso do computador na coleta e organização de dados para a produção de notícias. Mas a transformação digital e do mercado das notícias exigem novas habilidades do jornalista que não se resumem à literacia digital.

No artigo *How Computational Thinking is changing journalism & what's next?*, Kim Pearson (2009) defendeu a necessidade dos jornalistas dominarem os fundamentos do pensamento computacional. Um dos exemplos citados pela autora está relacionado

ao jornalismo de dados. Atualmente, as redações jornalísticas necessitam de profissionais com a capacidade de criar programas para a extração de dados (*data scraping*) de páginas da Web e de documentos, como PDFs, e transportá-los em um formato que facilite o processamento das informações. Isto requer não apenas conhecimentos de programação, mas sobretudo um conjunto de ferramentas cognitivas, como a capacidade de fazer abstrações, reconhecimento de padrões e outras habilidades relacionadas ao pensamento computacional.

Um bom exemplo da aplicação de métodos das Ciências da Computação para a produção de notícias vem do jornal britânico *The Guardian*. Em 2009, o jornal tornou público sua API (*Application Programming Interface*). Um dos objetivo era facilitar o acesso aos dados para que qualquer pessoa, principalmente outros jornalistas, pudessem integrá-los com outras aplicações da Internet (Anderson, 2009).

Nota-se que a habilidade de pensar computacionalmente é cada vez mais exigida por diferentes áreas de atuação profissional e científica. A ISTE e a CSTA elaboraram, em 2009, uma quadro (Quadro 1.1) que identifica conceitos centrais do pensamento computacional que podem ser aplicados em diversas disciplinas:

Quadro 1.1: Pensamento computacional como tema transversal do currículo

Conceitos e habilidades do Pensamento Computacional	Ciências da Computação	Matemática	Ciências	Estudos Sociais	Artes e linguagens
Coleta de dados	Encontrar fontes de dados para um problema da área	Encontrar fontes de dados para um problema da área, por exemplo, laçar moedas ou jogar dados	Coletar dados a partir de um experimento	Estatísticas de batalhas ou dados populacionais	Fazer análise linguística de frases
Análise de dados	Construir um programa para fazer cálculos estatísticos em um conjunto de dados	Contar ocorrências do lançamento de moedas e analisar os resultados (Estatística)	Analisar os dados de um experimento	Identificar tendências em dados estatísticos	Identificar padrões em diferentes tipos de frases
Interpretação de dados	Usar estrutura de dados, tais como listas, gráficos, etc.	Usar histogramas, gráficos circulares ou de barras para representar os dados	Resumir os dados de um experimento	Resumir e representar tendências	Representar padrões de diferentes tipos de sentenças

Decomposição de problemas	Definir objetos e métodos; definir principais componentes e funções	Aplicar a ordem de uma operação em uma expressão	Classificar espécies	-	Escrever um esboço
Abstração	Usar procedimentos para encapsular um conjunto de comandos frequentemente repetidos de uma função; usar condicionais, loops, etc.	Usar variáveis em Álgebra; identificar fatos essenciais em um problema; estudar funções em álgebra comparando às funções em programação; usar iteração para resolver problemas	Construir um modelo de uma entidade física	Resumir fatos; Deduzir conclusões a partir de fatos	Usar metáfora
Algoritmos e procedimentos	Estudar algoritmos clássicos; implementar um algoritmo para um problema da área	Fazer grandes divisões, trabalhar com fatoriais;	Fazer um procedimento experimental		Escrever instruções
Automação	-	Usar ferramentas como Star Logo; Python	Usar Probeware	Usar Excel	Usar um verificador de pronúncia
Paralelismo	Dividir dados e tarefas de forma que possam ser processados em paralelo; Usar a técnica de *pipeline*	Resolver sistemas lineares; fazer matrizes de multiplicação	Executar simultaneamente experimentos com diferentes parâmetros		
Simulação	Animação de algoritmo; varredura de parâmetro	Fazer um gráfico de uma função no plano cartesiano e modificar valores de variáveis	Simular movimentos do sistema solar	Jogar Age of Empires; Oregon Trail	Encenar uma história

Fonte: CSTA (2009). Computational Thinking Across the Curriculum. Disponível em: http://csta.acm.org/Curriculum/sub/CurrFiles/CTExamplesTable.pdf

1.4 Pensamento computacional e educação

O aumento da demanda por profissionais com a habilidade de pensar computacionalmente confirma a visão de Jeannete Wing (2006) de que o pensamento computacional será uma competência fundamental para todos os cidadãos do século XXI. Essa premissa é amplamente apoiada por pesquisadores das Ciências da Computação e da educação. Por isso, após os esforços para a elaboração de uma definição mais consistente do termo, os trabalhos mais relevantes dos últimos anos enfocam práticas pedagógicas e ferramentas digitais que ajudam a desenvolver o pensamento computacional desde o Ensino Básico ao superior.

Mark Guzdial (2008) considera que ensinar pensamento computacional para todas as pessoas requer abordagens diferentes daquelas que são usadas para formar profissionais da computação. O autor faz a seguinte ponderação a partir do ponto de vista de educadores da área:

> Desenvolver abordagens que serão aplicadas para todos os estudantes vai requerer de nós [professores de computação] responder a questões difíceis, como o que os estudantes de outras áreas entendem sobre computação, que desafios eles encontrarão, que tipos de ferramentas podem fazer o pensamento computacional mais facilmente acessível e como poderíamos organizar e estruturar nossas aulas para que a computação seja acessível para uma ampla variedade de estudantes. (p. 25, tradução livre)

Para Barr e Stepheson (2011), o desafio de incluir o pensamento computacional em programas curriculares é ainda maior quando se pensa em alunos do Ensino Básico. Os autores defendem uma abordagem mais prática que procura dar resposta aos seguintes questionamentos: "Como o pensamento computacional pode se apresentar em sala de aula? Que tipo de habilidades os estudantes poderiam demonstrar? O que é necessário para que um professor coloque em prática o pensamento computacional? Que tipo de práticas que já são implementadas pelos professores podem ser modificadas e estendidas?" (p. 50).

Essas perguntas vêm sendo respondidas por organizações profissionais, universidades e empresas da área da computação. A seguir, destacamos os trabalhos mais relevantes.

No contexto universitário, destaca-se o pioneirismo da Carnegie Mellon University que criou, em 2007, com apoio da Microsoft, o Center for Computational Thinking para apoiar pesquisas emergentes em Ciências da Computação, especialmente aquelas que ajudam a promover o pensamento computacional em outras disciplinas (Microsoft, 2007).

Outra iniciativa relevante da indústria tecnológica vem do Google, com o programa educativo *Exploring Computational Thinking*. Por meio desse projeto, o Google oferece uma coleção de planos de aula, vídeos e outros recursos para apoiar o ensino do pensamento computacional (Google, n.d.).

Com ênfase no público infantil, o projeto *Computational Thinking with Scratch* (http://scratched.gse.harvard.edu/ct/) apresenta conceitos, práticas e perspectivas do pensamento computacional que podem ser desenvolvidas por meio da utilização do Scratch. Os materiais disponíveis no *site* foram desenvolvidos por pesquisadores da Harvard Graduate School of Education, da EDC's Center for Children and Technology e do MIT Media Lab.

Nos Estados Unidos, destacam-se os esforços da CSTA e da ISTE, que têm trabalhado em conjunto no desenvolvimento de materiais para ajudar educadores a entender, valorizar e implementar o pensamento computacional no ensino básico e secundário (CSTA, 2011).

Também é importante ressaltar o trabalho do Conselho Nacional de Pesquisa dos Estados Unidos (NRC). A NRC realizou dois *workshops,* em 2009 e 2010, sobre o alcance, a natureza e o ensino do pensamento computacional que resultaram em dois importantes relatórios (National Research Council, 2009, 2011).

No Reino Unido, a academia científica britânica The Royal Society colaborou para a definição do pensamento computacional por meio de um estudo, publicado em 2012, sobre o ensino de fundamentos da computação em escolas do país. A organização Computing at School também é referência britânica na promoção e desenvolvimento de práticas pedagógicas que orientam a incorporação do pensamento computacional no currículo das escolas primárias.

Outra importante referência vem da Nova Zelândia. O projeto *CS Unplugged*, desenvolvido pelo departamento de Educação em Ciências da Computação da Universidade de Canterbury, propõe atividades lúdicas que estimulam o desenvolvimento do pensamento computacional sem a necessidade de computadores.

Os exemplos acima confirmam a previsão feita por Guzdial (2008) de que pesquisadores da área de educação em computação vão preparar o caminho para fazer do pensamento computacional uma literacia do século 21 transversal a todas as disciplinas.

1.5 Pensamento computacional no Ensino Básico

Como desenvolver o conjunto de competências necessárias para desenvolvimento do pensamento computacional no Ensino Básico? A CSTA elaborou diretrizes de currículo para o ensino de computação do Ensino Básico ao secundário que se transformaram em uma das principais referências pedagógicas nessa área. O documento *K–12 Computer Science Standards* apresenta objetivos de aprendizagens organizados em três níveis, agrupados por anos de escolaridade do sistema educativo americano. Conforme este modelo, o ensino da computação deve abordar: o pensamento computacional; a colaboração entre os estudantes; a prática da computação e programação; os computadores e dispositivos de comunicação; e os impactos éticos, globais e na comunidade (2011).

Destacamos os objetivos referentes à aprendizagem do pensamento computacional nos níveis correspondentes ao Ensino Básico a partir da versão resumida e traduzida do documento da CSTA publicada por Gresse Von Wangenheim et al. (2014):

Nível 1: "Ciência da Computação e Eu"

Este nível é recomendado para os estudantes do infantil até o sexto ano. Os alunos são introduzidos aos conceitos fundamentais em Ciências da Computação. Os educadores devem elaborar experiências de aprendizagem que ajudem os alunos a ver a computação como uma parte de seu mundo. Este nível está dividido em dois grupos: Nível 1A (Infantil ao 3º ano) e Nível 1B (3º ao 6º ano). Os objetivos de aprendizagem concernentes a cada grupo são descritos abaixo - Quadro 1.2 e Quadro 1.3.

Quadro 1.2 Objetivos de aprendizagem nível 1A (Infantil ao 3º ano)

1. Usar recursos tecnológicos (p.ex. quebra-cabeças, programas de pensamento lógico) para resolver problemas apropriados para a idade.
2. Usar ferramentas de escrita, câmeras digitais, e ferramentas de desenho para ilustrar pensamentos, ideias, e estórias de um modo passo a passo.
3. Entender como classificar/ordenar informação de modo útil, tais como estudantes classificados pela data de nascimento, sem o uso de um computador.
4. Reconhecer que software é criado para controlar operações do computador.
5. Demonstrar como 0s e 1s podem ser usados para representar informação.

Fonte: (CSTA, 2011); (Gresse Von Wangenheim et al. 2014).

Quadro 1.3 Objetivos de aprendizagem nível 1B (3º ao 6º ano)

1. Entender e usar os passos básicos para a solução de problemas algorítmicos (declaração e exploração do problema, identificação de exemplos, projeto, implementação e testes).
2. Desenvolver um entendimento simples de um algoritmo (busca, sequência de eventos ou ordenação/classificação) usando exercícios sem o uso de computador.
3. Demonstrar como uma cadeia de bits pode ser usada para representar informação alfanumérica.
4. Descrever como uma simulação pode ser usada para resolver um problema.
5. Fazer uma lista de sub-problemas para serem considerados enquanto resolve um problema maior.
6. Entender as conexões entre ciências da computação e outros campos.

Fonte: (CSTA, 2011); (Gresse Von Wangenheim et al. 2014).

Nível 2: "Ciência da Computação e Comunidade"

Neste nível, recomendado para os estudantes do sexto até o nono ano, os alunos começam a experimentar o pensamento computacional como ferramenta para resolução de problemas e uma forma de abordar questões relevantes para eles mesmo e para a comunidade. Os objetivos de aprendizagem são descritos no Quadro 1.4.

Quadro 1.4: Objetivos de aprendizagem Nível 2 (6. -9. ano – Ensino Fundamental II)

1. Usar os passos básicos de algoritmos para a resolução de problemas ao projetar soluções (p.ex., declaração e exploração do problema, exame de exemplos, design, implementação de uma solução, testes, avaliação).
2. Descrever o processo de paralelização na forma que se refere à resolução de problemas.
3. Definir um algoritmo, como sendo uma sequência de instruções que podem ser processadas por um computador.
4. Avaliar formas com que algoritmos diferentes podem ser utilizados para resolver o mesmo problema.
5. Dramatizar algoritmos de busca e ordenação.
6. Descrever e analisar uma sequência de instruções a ser seguida (p.ex., descrever o comportamento de um personagem em um videojogo, dirigido por regras e algoritmos).
7. Representar dados em maneiras diferentes, incluindo texto, sons, imagens e números.
8. Usar representações visuais de estados de problema, estruturas, e dados (p.ex., gráficos, tabelas, diagramas de rede, fluxogramas).
9. Interagir com modelos específicos de conteúdo e simulações (p.ex., ecossistemas, epidemias, dinâmica molecular) para apoiar a aprendizagem e pesquisa.

10. Avaliar que tipos de problemas podem ser resolvidos usando modelagem e simulação.
11. Analisar o grau em que um modelo de computador representa, com precisão, o mundo real.
12. Fazer uso da abstração para decompor um problema em subproblemas.
13. Compreender a noção de hierarquia e abstração em computação, incluindo linguagens de alto-nível, tradução (p. ex.,interpretar o mesmo problema de diferentes modos), conjunto de instruções, e circuitos lógicos.
14. Examinar conexões entre elementos da matemática e ciência da computação, incluindo números binários, lógica, conjuntos e funções.
15. Fornecer exemplos de aplicações interdisciplinares do pensamento computacional.

Fonte: (CSTA, 2011); (Gresse Von Wangenheim et al. 2014).

As diretrizes curriculares orientam o planejamento de experiências de aprendizagem do pensamento computacional. Os professores também podem planejar aulas a partir de recursos pedagógicos oferecidos por pesquisadores da área de educação em Ciências da Computação. Conhecer esses recursos também é imprescindível para o desenho de interfaces que ajudam os alunos a alcançar os objetivos de aprendizagem.

Diferentes ferramentas já são utilizadas com essa finalidade. Embora a maioria tenha como foco o ensino da programação, oferecem funcionalidades que facilitam o ensino e a aprendizagem de conceitos básicos do pensamento computacional, como abstração, pensamento algorítmico, decomposição, generalização, paralelismo e detecção sistemática de erros (Grover & Pea, 2013). Vejamos alguns exemplos de *softwares* educativos:

- Scratch (http://scratch.mit.edu/): ambiente de programação visual baseado em blocos e comunidade *online* desenvolvido pelo MIT Media LAB. É usado para introdução de crianças e jovens à programação e aos conceitos, práticas e perspectivas do pensamento computacional (Brennam & Resnick, 2012).
- MIT App Inventor (http://appinventor.mit.edu/): utiliza os mesmos fundamentos da programação visual do Scratch, mas o foco é a criação de aplicações para dispositivos móveis. O projeto é mantido pela equipe do Center for Mobile Learning do MIT.
- Alice (http://www.alice.org/): desenvolvido por pesquisadores da Carnegie Mellon University, é um ambiente de programação para iniciantes que permite a criação de animações em 3D e aprendizagem de conceitos básicos da computação.

- Kodu (http://snap.berkeley.edu/): linguagem de programação visual, desenvolvido pela Microsoft, que permite criar jogos, histórias e animações interativas.
- Blocky (https://blockly-games.appspot.com/): série de jogos educativos desenvolvidos pelo Google para o ensino de programação.

Outras ferramentas que podem ser usadas para ensino de pensamento computacional são os *kits* robóticos e interfaces tangíveis, como Arduino (https://www.arduino.cc/) (Grover & Pea, 2013).

Tal como justificado na Introdução deste trabalho, a nossa atenção será direcionada para o Scratch. No capítulo 4, faremos uma análise da ferramenta para entender como o seu ambiente de criação de projetos ajuda a desenvolver o pensamento computacional. O objetivo é entender as decisões que orientaram o desenho da sua interface que podem servir de referência para o desenvolvimento de novas interfaces de aprendizagem.

Capítulo 2: Pensar sobre aprendizagem

No capítulo anterior, vimos que a introdução do pensamento computacional no Ensino Básico deve ser guiada por objetivos de aprendizagem definidos em programas curriculares. Para desenhar sistemas interativos que ajudam a alcançar esses objetivos, os designers precisam compreender como as pessoas aprendem. Dorian Peters afirma, no livro *Interface Design for Learning: Design Strategies for the Learning Experience* (2013), que as "teorias fornecem explicações de como e por que fazemos algo, e para criar interfaces para a aprendizagem que são verdadeiramente eficazes, precisamos de uma compreensão básica de como as pessoas aprendem coisas novas" (p. 11).

O objetivo deste capítulo é apresentar os fundamentos das principais teorias da aprendizagem e as influências dessas teorias no desenvolvimento de tecnologias educativas.

2.1 Teorias da aprendizagem

Durante o século XX, houve um crescimento de pesquisas que deram origem a grandes teorias da aprendizagem que ajudaram a legitimar políticas educacionais, o papel dos professores e dos estudantes, a escolha e apresentação de materiais educativos e o desenvolvimento de tecnologias educativas. Nas décadas de 50 e 60, prevaleceram os estudos com foco na modificação do comportamento fundamentados pela teoria behaviorista. Nas duas décadas seguintes, 70 e 80, prevaleceu o interesse sobre o desenvolvimento de habilidades cognitivas a partir de estudos apoiados em teorias construtivistas e cognitivistas (Selander, 2016).

Dessas três grandes teorias resultaram diferentes perspectivas sobre ensino e aprendizagem e sobre o uso de tecnologias na educação. Elas também deram origem a outras teorias que propõem novas concepções de aprendizagem no contexto multimídia e conectado por tecnologias digitais. Apresentamos a seguir as premissas e os principais teóricos e investigadores dos diferentes paradigmas da aprendizagem.

2.1.1 Behaviorismo

Behaviorismo é uma corrente da psicologia que se focaliza nos aspectos objetivos e observáveis do comportamento (*behavior*, em inglês). A teoria behaviorista fundamenta-se na ideia de que a aprendizagem ocorre quando existe mudança de comportamento. A mudança é resultado da resposta individual aos estímulos fornecidos

pelo ambiente externo. Para os behavioristas, o que ocorre na mente durante o processo de aprendizagem não pode ser objetivamente observável e quantificado, por isso não abordam conceitos como consciência, emoção e memória. O que interessa, para os adeptos dessa corrente, é a análise das dimensões observáveis e mensuráveis do comportamento e o mecanismo de "estímulo-resposta".

Dois dos principais teóricos do behaviorismo foram os psicólogos John B. Watson e Burrhus Frederic Skinner. O termo behaviorismo aparece pela primeira vez em 1913 no artigo *Psychology as the behaviorist views it*, assinado por Watson. O autor, que é referenciado como behaviorista clássico, defendia a observação objetiva das respostas do organismo a estímulos controlados.

Skinner é considerado o principal representante do behaviorismo radical. Ele desenvolveu a teoria do condicionamento operante segundo a qual o comportamento é determinado por suas consequências positivas ou negativas, que podem influenciar a probabilidade de este ocorrer novamente. O condicionamento operante é um mecanismo que premia uma determinada resposta de um indivíduo até ele ficar condicionado a associar a necessidade à ação (Ferrari, 2011).

Outro conceito-chave no trabalho de Skinner é a ideia de reforço, definido como qualquer acontecimento (estímulo) que segue uma resposta e aumenta a probabilidade dessa resposta ocorrer no futuro. O reforço pode ser positivo, quando adicionam um estímulo agradável após um comportamento desejado, ou negativo, quando eliminam um evento desagradável para promover um comportamento desejado. Ambos podem ser entendidos como recompensas (Peters, 2013).

Enquanto o objetivo do reforço é aumentar a probabilidade de repetição de um comportamento, a punição tem como finalidade a extinção do comportamento. A punição também tem duas categorias: punição positiva, que se refere à adição de experiências desagradáveis após a realização de um comportamento não desejado; e a punição negativa, entendida como a remoção de uma experiência agradável após a realização de um comportamento não desejado.

Skinner propôs a aplicação do princípio do reforço positivo na educação (Ferrari, 2011). Os seus estudos, assim como os trabalhos de outros pesquisadores da corrente behaviorista, serviram como base teórica para o desenvolvimento de máquinas de ensino, objetivos de aprendizagem mensuráveis, instrução assistida por computador (*Computer-Assisted Instruction* – CAI) e testes de múltipla escolha (Bates, 2015).

É importante ressaltar que o aprendiz tem um papel passivo na perspectiva behaviorista da aprendizagem. A premissa fundamental é que seu comportamento pode ser moldado por reforços positivos ou negativos.

2.1.2 Construtivismo

A teoria construtivista fundamenta-se na visão do indivíduo como construtor de conhecimento a partir de suas experiências. A aprendizagem é um processo de ajuste de nossos modelos mentais para dar sentido a novas experiências. Enquanto o behaviorismo ignora a investigação sobre a mente humana por considerá-la uma "caixa negra", o construtivismo e outras teorias cognitivas enfatizam os processos mentais. Ao olhar para esses processos, os teóricos construtivistas concentram-se na percepção, interpretação e construção individual (Guttormsen, 2010).

Os principais teóricos associados ao construtivismo são Jean Piaget, Jerome Bruner e Lev Vygotsky.

a) Piaget

O suíço Jean Piaget é considerado um dos investigadores mais influentes do século 20. Além de ter lançado as bases para os campos da psicologia do desenvolvimento, da teoria cognitiva e da epistemologia evolutiva, foi um dos primeiros teóricos a levar a sério o pensamento das crianças (Papert, 1999). Seus estudos evidenciaram que as crianças têm suas próprias visões de mundo, diferente das visões dos adultos, mas ainda assim extremamente coerentes e robustas (Ackermann, 2004).

Em suas pesquisas sobre a gênese do conhecimento (epistemologia genética), o objetivo de Piaget é explicar o processo de desenvolvimento intelectual do indivíduo enquanto sujeito epistêmico. Para Piaget, o conhecimento cresce de acordo com complexas leis de auto-organização (Ackermann, 2004). As principais contribuições da teoria de Piaget é a descrição do mecanismo de equilibração e a identificação de quatro estágios do desenvolvimento cognitivo pelos quais as crianças passam.

A teoria da equilibração descreve o que ocorre no sistema cognitivo durante a relação entre o sujeito e o ambiente externo. Ela está subordinada a duas propriedades: a organização e adaptação.

A organização é a tendência para ajustar o sistema cognitivo às mudanças para manter sua estabilidade. Já a adaptação consiste em dois processos: a assimilação, que é a integração de um novo conhecimento às estruturas cognitivas já existentes; e a

acomodação, que é a formação de novas estruturas quando não é possível assimilar o novo conhecimento às estruturas cognitivas prévias. Enquanto a assimilação incorpora um acontecimento do ambiente, conservando a organização anterior, a acomodação modifica as estruturas que já estavam organizadas para que o indivíduo se ajuste em função das estruturas do acontecimento (Ferreira, 2003). Esse processo progride em níveis crescentes de complexidade.

Os estudos de Piaget identificam quatro estágios do desenvolvimento cognitivo:

- Sensório-motor (0 - 2 anos)

Nesta fase, a criança aprende por meio do contato direto com o ambiente que a cerca. O desenvolvimento cognitivo é baseado nas sensações e nos movimentos. É nesta etapa que a criança adquire a noção de permanência do objeto.

- Pré-operatório (2 – 7 anos)

A criança torna-se capaz de pensar simbolicamente e usar a linguagem, mas não é capaz de perceber o ponto de vista de outros (egocentrismo intelectual). O pensamento é intuitivo e não realiza operações lógicas.

- Operações concretas (7 - 11 anos)

A criança desenvolve o pensamento lógico e torna-se capaz de realizar operações mentais, como adição, subtração e as conservações de número, substancia, volume e peso. Desenvolve também noções de tempo, espaço, velocidade, ordem, casualidade.

- Operações formais (11 – 15 anos)

No período formal as crianças desenvolvem o pensamento abstrato e o raciocínio hipotético-dedutivo. É a partir desta fase que o indivíduo pode alcançar a maturidade intelectual.

O trabalho de Piaget foi amplamente divulgado entre as décadas de 50 e 70 e influenciou reformas educacionais nesse período, embora esse não fosse o objetivo do autor (Papert, 1999). Ackermann (2004) destaca três implicações da teoria de Piaget para a educação:

1. O ensino não pode ser direto: crianças não apenas absorvem o que é dito. Em vez disso, elas interpretam o que escutam à luz dos seus conhecimentos e experiências.

2. O conhecimento não é uma informação a ser levada de uma extremidade, codificada, armazenada, recuperada e reaplicada na outra extremidade.
3. A teoria da aprendizagem não pode ignorar as resistências à aprendizagem.

b) Bruner

A visão de aprendizado subjacente às pesquisas de Jerome Bruner é a de um processo ativo no qual os aprendizes constroem novas ideias e conceitos a partir do conhecimento que eles já possuem. O autor descreve etapas do desenvolvimento cognitivo que correspondem a três modos de representação da interação com o mundo:

- Inativo - baseado na ação. Neste modo, a interação com o mundo é feita por meio de toques, manipulação e ação sobre os objetos.
- Icônico – baseado em imagens, consiste na representação visual da realidade.
- Simbólico – baseado na linguagem como forma de representação da realidade.

Bruner propõe que os métodos de instrução podem ser adaptados aos diferentes modos de ver o mundo. Ele também defende a aprendizagem por descoberta em que destaca o papel ativo do aprendiz na construção do seu próprio conhecimento. Aprendizagem por descoberta ou exploração acontece quando os aprendizes são estimulados pelo ambiente de aprendizagem para a resolução de problemas, a aprender de forma independente por meio de suas experiências, entendidas como oportunidades para o desenvolvimento de novas perspectivas sobre questões e temas complexos (Guttormsen, 2010).

Ao descrever o papel do instrutor nesse processo de descoberta, Wood, Bruner e Ross (1976) utilizam a metáfora do andaime – *scaffolding*, que consiste essencialmente no controle do adulto sobre os elementos da tarefa que estão inicialmente além da capacidade do aprendiz, permitindo que este se concentre e complete apenas os elementos que estão ao alcance de suas competências.

c)Vygotsky

Vygotsky enfatizou a função das interações sociais no processo de aprendizagem e elaborou os referenciais teóricos fundamentais para o sócio-construtivismo. Seu trabalho foi movido pelo interesse de investigar o papel do ambiente social, com suas ferramentas e seus objetos culturais, e das pessoas como agentes do desenvolvimento intelectual (Bransford et al, 2000).

O sócio-construtivismo de Vygotsky salienta a importância dos adultos, da linguagem e de outros artefatos culturais no processo de aprendizagem. Nesta perspectiva, a criança aprende internalizando atividades, hábitos, vocabulários e ideias dos membros da comunidade em que está inserida. Entretanto, apesar de seu foco na importância da cultura para a aprendizagem, Vygotsky viu o desenvolvimento intelectual da criança como um processo construtivo.

Um dos conceitos-chave da teoria de Vygotsky é a ideia de Zona de Desenvolvimento Proximal, que corresponde à área de potencial expansão do desenvolvimento cognitivo que cada indivíduo tem à sua disposição quando ajudado por outros (Ackermann, 2004).

Para Vigostsky, o desenvolvimento da criança ocorre de duas formas: primeiro, no nível interpsicológico, e depois no nível intrapsicológico. Nas palavras de Ackermann (2004), "relações interpessoais são precursoras e condições necessárias para a emergência dos processos mentais de cada indivíduo" (p. 22).

2.1.3 Construcionismo

O mais importante mentor intelectual do construcionismo foi Seymour Papert. Após estudar desenvolvimento cognitivista com Piaget, Papert ajudou a fundar o Laboratório de Inteligência Artificial do Instituto Tecnológico de Massachusetts – MIT. Foi nesse ambiente que desenvolveu, em 1967, a primeira versão da linguagem de programação Logo, desenhada especialmente para crianças.

A experiência com o Logo foi fundamental para o desenvolvimento da teoria construcionista nas décadas seguintes. Para explicá-la, é necessário fazer uma distinção entre a teoria de Papert e o construtivismo. Ambas compartilham a noção de aprendizado como construção de estruturas de conhecimento, independente das circunstâncias de aprendizagem. Porém, o construcionismo acrescenta a ideia de que o aprendizado será facilitado especialmente em um contexto em que o aprendiz esteja conscientemente engajado na construção de algo, "seja um castelo de areia na praia ou uma teoria do universo" (Papert, 1991, p. 1). A valorização do envolvimento dos aprendizes em atividades com as quais se identificam é, portanto, um princípio fundamental nesta teoria.

Segundo Papert (1991), a definição mais simples de construcionismo evoca a ideia de aprender fazendo (*learning by making*). A abordagem construcionista vê o aprendizado como construção de relações entre o velho e o novo conhecimento por meio da criação de artefatos socialmente relevantes (Kafai, 2006). Nesta perspectiva, as novas tecnologias têm a vantagem de expandir o leque de atividades que podem ser realizadas pelos aprendizes. Softwares educativos podem ser "*objects-to-think-with*" que ajudam as crianças a refletir sobre suas próprias performances. Papert dedicou a maior parte de sua vida criando ferramentas digitais para crianças a partir de sua concepção de micromundo, ambientes de aprendizagem em que os aprendizes podem explorar, descobrir e simular diferentes conceitos e ideias de forma segura (Papert, 1980; Ackermann, 2004).

O construcionismo também valoriza a ideia de cultura de aprendizagem (*learning cultures*) . "Embora os pesquisadores socioculturais enfatizem as dinâmicas sociais das culturas de aprendizagem, os construcionistas concentram-se em como o contexto social oferece oportunidades para fazer conexões com o que está sendo aprendido" (Kafai, 2006, p. 40).

As principais investigações que ajudaram a formar o corpo teórico do construcionismo foram elaboradas por pesquisadores do grupo de Epistemologia e Aprendizagem (Epistemology and Learning Group), do MIT, fundado por Papert em 1981. Blikstein ressalta a importância dos trabalhos desenvolvidos pelos pesquisadores desse grupo para a nova geração de educadores e para o atual movimento *maker:*

> O Grupo de Epistemologia e Aprendizagem, do MIT, que Papert fundou em 1981, atraiu uma legião de brilhantes estudantes e pesquisadores que, durante as próximas duas décadas, trariam a milhões de crianças ideias e tecnologias avançadas, tais como robótica, modelagem multi-agente, sistemas dinâmicos e fabricação digital. A própria empresa Lego acabaria por transformar muitas dessas ideias em produtos sob o título de "Mindstorms", em homenagem a Papert" (2014, para. 6, tradução livre).

2.1.4 Cognitivismo

Do latin *cognitio*, que significa conhecimento, o cognitivismo investiga os processos mentais (memória, percepção e raciocínio) implicados na mudança de comportamento. Contrapondo-se ao behaviorismo, a abordagem cognitivista busca entender o funcionamento da mente. Como observa Ng, "pesquidadores cognitivistas investigam como a mente recebe, organiza, armazena e recupera a informação" (2015, p.86).

A descrição desse processo resulta na elaboração de estratégias de aprendizado para que os aprendizes possam otimizar o processamento da informação. Ertmer e Newby (2013) afirmam que o verdadeiro foco da abordagem cognitivista é estimular o aprendiz a usar as melhores estratégias de aprendizagem. A propósito dessa característica, os autores explicam o papel dos professores e designers instrucionais:

> Aprendizagem ocorre quando a informação é armazenada na memória de forma organizada e significativa. Professores/designers instrucionais são responsáveis por ajudar os aprendizes na organização dessa informação da melhor forma. Designers usam técnicas, como organizadores prévios, analogias, relações hierárquicas e matrizes, para ajudar os aprendizes a relacionar novas informações aos seus conhecimentos prévios. (p. 52, tradução livre)

O desenvolvimento do campo das Ciências da Computação e da Inteligência Artificial influenciou a abordagem cognitivista que entende a mente humana como processador de informação análogo aos computadores. Segundo esse modelo, o processamento da informação possui três componentes: memória sensorial, memória de curta duração e memória de longa duração. "Estímulos externos e a informação detectada pelos olhos (visual) e ouvidos (sonora) passam pelos canais sensoriais para a memória de curta duração, onde a informação é processada, para então ser codificada e armazenada de forma esquemática na memória de longa duração" (Ng, 2015, p.86).

Nota-se o papel proeminente da memória na abordagem cognitivista. Outro conceito importante é o de esquema (*schema*), que se refere às estruturas cognitivas ou quadro de referência mental que utilizamos para comparar e organizar novas informações. Mas como a ideia de esquema influencia na aprendizagem?

> Cognitivistas afirmam que a aprendizagem funciona melhor quando podemos conectar nova informação a fatos que já conhecemos. (..) Esquemas fornecem uma estrutura à qual podemos anexar nova informação. Esquemas são dinâmicos e mudam assim que interpretamos novas experiências e adaptamos nosso entendimento de forma apropriada. (Peters, 2013, p. 21, tradução livre)

Os pesquisadores cognitivistas investigam as limitações no processo cognitivo humano e as formas de superá-las para aumentar a capacidade de processamento de informação. Um dos principais resultados dessas pesquisas é a teoria da Carga Cognitiva (*Cognitive Load*), desenvolvida por John Sweller, que, resumidamente, refere-se à quantidade de informação que um cérebro humano pode processar a cada momento. Conforme esta teoria, há três tipos de carga cognitiva com influência na aprendizagem:

a) Intrínseca: é determinada pelo nível de complexidade inerente aos materiais instrucionais que devem ser processados simultaneamente na memória de curta duração do aprendiz. A quantidade de elementos que pode ser processada depende do nível de especialização do aprendiz e do grau de dificuldade da tarefa. Educadores e designers instrucionais não podem controlar a carga cognitiva intrínseca, mas devem compreendê-la para garantir que os materiais de aprendizagem tenham número apropriado de elementos para garantir a aprendizagem (Ng, 2015).

b) Extrínseca: é determinada pela forma de apresentação dos materiais instrucionais. Elementos visuais ou auditivos usados de forma inadequada podem sobrecarregar a memória de curta duração do aprendiz, dificultando sua aprendizagem. Educadores e designers instrucionais podem controlar a carga cognitiva extrínseca.

c) Pertinente: é determinada pelo desenho de materiais instrucionais que fomentam o processo de aprendizagem. Esta carga também está sob controle de educadores e designers instrucionais.

Desta forma, boas estratégias pedagógicas e bons ambientes de aprendizagem resultam da redução da carga cognitiva extrínseca e do aumento da carga cognitiva pertinente.

2.1.5 Conectivismo

O conectivismo parte do princípio de que as mudanças promovidas pela era digital implicam em novas necessidades e processos de aprendizagem que não são explicados pelas teorias behavioristas, cognitivistas e construtivistas.

E m *Connectivism: A Learning Theory for the Digital Age*, George Siemens (2005) propõe uma teoria da aprendizagem que, segundo sua avaliação, seria mais adequada para o contexto atual, em que o conhecimento cresce e torna-se obsoleto a um ritmo sem precedentes. Aprender, neste cenário, consiste em conectar conjuntos de informações especializados. As conexões mais importantes são aquelas que nos capacitam a aprender mais e de forma contínua.

No mesmo artigo, o autor apresenta nove princípios do conectivismo:

• Aprendizagem e conhecimento apoiam-se na diversidade de opiniões.

• Aprendizagem é um processo de conexão entre nós especializados ou fontes de informação.

• Aprendizagem pode residir em dispositivos não humanos.

• A capacidade de saber mais é mais crítica do que aquilo que é conhecido atualmente.

• É necessário cultivar e manter conexões para facilitar a aprendizagem contínua.

• A habilidade de enxergar conexões entre áreas, idéias e conceitos é uma habilidade fundamental.

• A atualização constante do conhecimento é a intenção de todas as atividades de aprendizagem conectivistas.

• A tomada de decisão é, por si só, um processo de aprendizagem. Escolher o que aprender e o significado das informações que chegam é enxergar através das lentes de uma realidade em mudança.

O conectivismo também aborda a distribuição do conhecimento em ambiente organizacional e a análise de redes sociais para compreensão dos modelos de aprendizagem na era digital.

2.2 Tecnologia e aprendizagem

As teorias que explicam como ocorre o processo de aprendizagem, resumidas acima, também legitimam os diferentes formatos e grau de importância do uso de tecnologias na educação, sobretudo no que se refere aos computadores. Koschmann (1996) identifica quatro paradigmas que marcam a evolução do uso do computador na educação:

- CAI (Computer Assisted Instruction): Surge em 1960 com o lançamento do Coursewriter I da IBM, que é considerado o primeiro software de autoria. A visão de aprendizagem predominante neste paradigma é de um processo passivo de aquisição de conhecimento. O modelo de instrução é baseado na ideia de transmissão de informações. A instrução assistida por computador fundamenta-se teoricamente no behaviorismo.
- ITS (Intelligent Tutoring System): Aparece no início da década de 70 e está diretamente associado à migração de pesquisadores da área de inteligência

artificial para a área educacional. O paradigma fundamenta-se na ideia de que a aprendizagem poderia ser melhorada se cada estudante tivesse um tutor pessoal. Os sistemas tutoriais inteligentes aplicam os conceitos desenvolvidos por pesquisas em inteligência artificial e cognitivismo.

- Logo-as-Latin: Surge na década de 1980 com a publicação de *Mindstorms: Children, Computers, And Powerful Ideas,* por Seymour Papert. O computador deixa de ser um "tutor", como nos paradigmas anteriores. O aprendiz assume um papel mais ativo na medida em que se envolve em atividades de programação. Segundo Koschmann, o construtivismo é a teoria da aprendizagem subjacente a esse paradigma.
- CSCL (Computer Supported Collaborative Learning): Os eventos que marcam o início deste paradigma foram o workshop *Special Program on Advanced Educacionl Tecnology*, em 1989, e o workshop on *Computer Support for Collaborative Learning* realizado entre 4 e 6 de Outubro de 1991 na Southern Illinois University. O objetivo é o desenvolvimento de recursos informáticos para apoiar a aprendizagem em grupo e tem como fundamentação as teorias que ressaltam as dimensões sociais da aprendizagem.

É importante notar que esses diferentes paradigmas ainda estão ativos no vasto leque de softwares educativos. A adoção dessas referências em ambientes educacionais depende do contexto, do conteúdo, do público e do objetivos de aprendizagem.

No próximo capítulo, abordamos os fundamentos que devem orientar o desenho de softwares educativos, em geral, e os de promoção do pensamento computacional, em particular.

Capítulo 3: Recomendações de design para softwares educativos

O desenho de interfaces de softwares educativos para o desenvolvimento do pensamento computacional em estudantes do Ensino Básico deve ser orientado por princípios consolidados em importantes estudos na área da Interação Homem Computador (*Human Computer Interaction – HCI*).

Segundo o SICGHI (*Special Interest Group on Computer–Human Interaction*), grupo de trabalho da ACM (*Association for Computing Machinery*), a Interação Homem Computador é a disciplina que estuda o *design*, a avaliação e a implementação de sistemas computacionais interativos para a utilização humana, assim como os fenômenos principais que acompanham esta utilização – a interação (Fonseca, Campos & Gonçalves, 2012).

Umas das principais contribuições dos investigadores no campo da Interação Homem-Computador foi o desenvolvimento de métodos para a promoção da usabilidade dos sistemas interativos. Bem aplicados, os critérios facilitam a interação com determinada interface e reduzem as possibilidades de não aceitação dos sistemas por parte dos usuários. Porém, quando se trata do desenho de softwares educativos, não basta seguir esses critérios.

Interfaces de aprendizagem não são desenhadas apenas para que os usuários executem uma tarefa, mas sim para que aprendam algum assunto ou desenvolvam alguma habilidade enquanto realizam a interação. Em que medida as decisões de design da interface podem ajudar no processo de aprendizagem?

Para responder a essa questão, é necessário uma abordagem que avance do Desenho Centrado no Usuário para o Desenho Centrado no Aprendiz (Soloway, Guzdial & Hay, 1994). Quando o aprendiz é um estudante de ensino básico, que abrange a faixa etária de 6 a 14 anos, é preciso considerar também as peculiaridades da interação entre a criança e o computador. Além disso, o desenho da interface deve ser uma expressão do tipo de conteúdo ou habilidade que os aprendizes estão tentanto aprender (Quintana et al, 2000).

Neste capítulo, apresentamos conceitos e recomendações para o desenho de interface de aprendizagem elaborados em pesquisas sobre Desenho Centrado no Aprendiz e sobre Interação Criança-Computador. A terceira seção apresenta recomendações e boas práticas do desenho de interfaces que promovem o pensamento computacional.

Optamos pela separação das recomendações de *design* concernentes aos três tópicos abordados: aprendizagem, criança e pensamento computacional. No entanto, observaremos que as fronteiras entre elas são difusas porque, apesar de terem pontos de partida diferentes, o objetivo é o mesmo: desenhar interfaces que efetivamente ajudem a desenvolver habilidades cognitivas nos usuários.

3.1 Princípios da usabilidade

Os componentes fundamentais dos sistemas interativos são o usuário, o sistema em si e a natureza do processo de interação (Dix, Finlay, Abowd & Beale, 2004, p. 11). O desenho da interface é crucial para a efetiva interação e deve ser concebido para ir ao encontro das necessidades dos usuários. Estudos na área da Interação Homem-Computador utilizam conceitos de diversos campos, como Ciências da Computação, Psicologia e Design, para fornecer métodos para a construção de interfaces mais acessíveis, eficientes e fáceis de usar, coerentes com os princípios gerais da usabilidade.

Segundo Jacob Nielsen (2012), "usabilidade é um atributo de qualidade que avalia a facilidade de uso de uma interface" (para.1). O autor ressalta que a palavra também "se refere ao conjunto de métodos usados para melhorar a facilidade de uso durante o processo do desenho da interface" (para.1).

Benyon (2013, p. 117) destaca as seguintes características de um sistema com alto grau de usabilidade:

- Será eficiente na medida em que as pessoas serão capazes de realizar as tarefas necessárias, usando uma quantidade adequada de esforço.
- Será eficaz na medida em que contém funções e informações adequadas e organizadas de maneira correta.
- Será fácil de aprender como fazer as coisas e lembrar de como fazê-las após um certo tempo.
- Será seguro de operar na variedade de contextos de uso.
- Terá grande utilidade na medida em que faz as coisas que as pessoas querem que seja feito.

Estes itens vão ao encontro dos cinco parâmetros de qualidade da usabilidade propostos por Nielsen (2012):

- Fácil de aprender: os usuários conseguem realizar tarefas básicas logo no primeiro contato que têm com a interface;
- Eficiência: assim que aprende como funciona o desenho da interface, o usuário consegue realizar as tarefas rapidamente;
- Fácil de lembrar: quando o usuário volta a utilizar a interface depois de um certo período, não terá dificuldade em lembrar como interagir com o sistema;
- Prevenção de erros: minimiza a possibilidade de erros, avisa o usuário e permite que ele consiga corrigir erros cometidos;
- Satisfação: a interação com o sistema é agradável.

Nielsen também propôs 10 princípios gerais para o bom design de sistemas interativos. Essas regras, conhecidas como heurísticas de usabilidade, são usadas para orientar o desenho e avaliação de interfaces.

1) Visibilidade do estado do sistema: manter o usuário informado sobre o que está acontecendo com o sistema por meio de feedback adequado e no tempo certo.
2) Correspondência entre o sistema e o mundo real: os conceitos, termos, tarefas e procedimentos adotados na interface devem ser familiares ao usuário e corresponder ao domínio de atividade a que o sistema se refere.
3) Controle e liberdade do usuário: a interface deve permitir que o usuário controle o sistema e oferecer alternativas para que ele possa anular e refazer uma operação.
4) Consistência e padrões: manter as convenções da plataforma em que o sistema é executado.
5) Prevenção de erros: reduzir as condições passíveis de erro dos usuários, informando-o sobre as consequências de uma determinada ação.
6) Reconhecimento ao invés de recordação: ajudar os usuários a reconhecer os componentes da interface. É importante que os objetos, ações e opções estejam visíveis para que o usuário identifique rapidamente as ações que precisa exeecutar.

7) Flexibilidade e eficiência de uso: ofereçer diferentes formas de acionar uma ação e permitir aos usuários personalizar ações frequentes. Dessa forma, a interface contemplará os usuários inesperientes e experientes.
8) Estética e Design Minimalista: evitar informações irrelevantes.
9) Ajudar os usuários a reconhecer, diagnosticar e corrigir erros: o sistema deve ter mensagens de erros claras, que informem o problem de forma precisa e proponha uma solução.
10) Ajudas e documentação: ofereça ajuda ao usuário em todas as ações.

As heurísticas de Nielsen são genéricas e úteis para análise de interface de qualquer sistema interativo do ponto de vista de sua usabilidade. No entanto, no que se refere às interfaces de aprendizagem, outros fatores devem considerados a partir da abordagem do Desenho Centrado no Aprendiz.

3.2 Desenho Centrado no Aprendiz

O desenvolvimento de interfaces baseado em Desenho Centrado no Usuário (DCU) caracteriza-se por colocar o usuário em primeiro plano, incluindo-o no processo do design, para compreender suas necessidades e as atividades que serão realizadas por meio do sistema interativo. Soloway et al. (1994) consideravam que essa metodologia não contemplava todos os desafios do desenho de interfaces de softwares educativos e propuseram uma abordagem de desenho centrado no aprendiz, que prioriza as necessidades de quem precisa desenvolver perícia em um domínio de atividade.

É necessário uma abordagem diferenciada para o desenho de interfaces de aprendizagem devido às características específicas da audiência e do objetivo da interação. Para Quintana, Soloway e Krajcik (2000), os aprendizes se diferenciam da noção mais genérica de usuários em diversos aspectos, entre os quais o nível de conhecimento, objetivo da interação, diversidade dos perfis e motivação. Os autores elaboram um quadro comparativo (Quadro 3.1) entre a tradicional abordagem de desenho centrado no usuário proposta por Don Norman (2013) e o Desenho Centrado no Aprendiz.

Em termos de audiência, o desenho centrado no usuário pressupõe que o usuário pode não saber como interagir com a interface, mas tem conhecimento sobre o tipo de tarefa que precisa realizar, pertence a um grupo mais homogêneo, tem intrínseca

motivação para a interação e não necessita de adaptações na interface. Portanto, o objetivo do designer é desenhar uma interface para que o usuário realize seu trabalho de forma mais fácil e eficiente.

Quadro 3.1: Diferenças entre usuários e aprendizes

Aspectos do design	Desenho para usuários	Desenho para aprendizes
Objetivo primário	Desenhar ferramentas computacionais com alta usabilidade que permita aos usuários as tarefas de forma fácil e eficiente	Desenhar ferramentas computacionais que ajudem os aprendizes a desenvolver um entendimento de uma prática profissional (sem ignorar a usabilidade)
Perfil da audiência	Usuário	Aprendiz
	Tem conhecimento sobre o domínio de atividade que será realizada por meio das ferramentas; Formam uma audiência mais homogênea porque compartilham a mesma cultura de trabalho; Possuem mais motivação intrínseca; Não terão significativa mudanças enquanto realizam as tarefas.	Tem um conhecimento incompleto ou mínimo sobre o domínio de atividade; Formam uma audiência heterogênea porque não possuem uma cultura de trabalho em comum; Possuem menos motivação intrínseca; Apresentação uma mudança significativa na medida em que aprendem sobre a atividade, o que implica a necessidade de mudanças na ferramenta.
Lacunas conceituais a abordar	Golfos de execução e avaliação entre o usuário e a ferramenta	Golfo de aprendizagem e o domínio da atividade a ser aprendida (sem ignorar os golfos existentes entre o usuário e a ferramenta)
Abordagem subjacente para fazer a travessia entre os golfos	Usa a teoria da ação que descreve como as pessoas usam as ferramentas para completar uma tarefa para que os designers possam minimizar os golfos de execução e de avaliação.	Usa teoria da aprendizagem para ajudar o aprendiz a atravessar o golfo de aprendizagem

Fonte: Quintana et al. (2000, 2003)

A usabilidade também é importante para o desenho centrado no aprendiz, mas não é suficiente para que ele realize a tarefa já que não possui o mesmo domínio do conteúdo e habilidades dos usuários experientes. Além disso, os aprendizes são mais heterogêneos porque nem sempre compartilham a mesma cultura de trabalho, linguagem ou conhecimento. Ao contrário dos usuários especialistas, os aprendizes não

estão intrisicamente motivados, já que encontram mais dificuldades na realização da tarefa. Por fim, os aprendizes necessitam de adaptações na interface na medida em que evoluem na compreensão do assunto.

Quintana et al. (2000) também afirmam que cada metodologia de desenho de interface lida com problemas diferentes. Os autores utilizam a metáfora do golfo proposta por Norman no livro *Design of Everyday Things* (2013), para comparar o desafio central de cada paradigma.

Nessa perspectiva, as pessoas utilizam um produto ou uma interface de um sistema interativo com um objetivo em mente. Norman (2013) descreve sete estágios da ação do usuário para alcançar o seu objetivo, que podem ser classificados como etapas de execução e avaliação:

Etapas de execução:

- Estabelecimento da meta;
- Determinação da intenção;
- Especificação da ação (sequência de ações necessárias para o objetivo);

Execução da ação;

- Etapas da execução:
- Percepção do estado do sistema;
- Interpretação do estado do sistema;
- Avaliação da resposta em relação à meta.

Entre o objetivo do usuário e estado do sistema há discrepâncias ou golfos. O golfo de execução consiste na diferença entre o objetivo do usuário e as ações permitidas pela ferramenta. O golfo de avaliação corresponde à quantidade de esforço que o usuário deve exercer para interpretar o estado da ferramenta após sua ação. Cabe ao designer, por meio da abordagem do Desenho Centrado no Usuário, desenvolver interfaces que façam a ponte entre o usuário e o sistema diminuindo as distâncias entre estes golfos.

O Desenho Centrado no Aprendiz considera a existência de um terceiro golfo: o de aprendizagem. A interface deve apoiar o aprendiz para que ele alcance o domínio do assunto que precisa aprender. "Como os aprendizes precisam desenvolver uma compreensão sobre um determinado domínio de atividade, uma ferramenta centrada no aprendiz precisa apoiá-lo na travessia do golfo de expertise entre o aprendiz e o domínio de atividade" (Quintana et al., 2000, p. 258).

Cada paradigma do desenho de interface fundamenta-se em diferentes teorias para resolver os problemas apresentados anteriormente. No Desenho Centrado no Usuário, a "a teoria da ação" proposta por Norman ajuda o designer a entender a interação das pessoas com as ferramentas. A partir desse modelo, surgem um conjunto de princípios para o desenho de uma interface eficiente e fácil de usar.

Já no Desenho Centrado no Aprendiz, o desenho do sistema deve refletir o assunto ou habilidades que os aprendizes estão tentanto dominar e deve ser guiado por um modelo de aprendizagem que facilite a construção do novo conhecimento. Por conta disso, Quintana et al. (2000) sugerem a contribuição de educadores no processo do design e a incorporação de teorias da aprendizagem para fundamentar as decisões. Para os autores, os designers precisam, antes de tudo, entender como as pessoas aprendem para então desenvolver software centrado no aprendiz. Por isso, as ciências da aprendizagem devem ser integradas no processo do design do software.

Os paradigmas das tecnologias educativas (Koschmann, 1996), abordados no capítulo anterior, demonstram que essa integração resultou em diferentes ambientes de aprendizagem computacionais.

Mais recentemente, Quintana et al. (2003) voltam ao assunto para descrever os diferentes estilos de condução do processo de design, implementação e avaliação de software educativo conforme a teoria de aprendizagem subjacente e o contexto em que o software será usado.

Em síntese, o Desenho Centrado no Aprendiz foca em um perfil de usuários que necessitam de apoio e motivação adicionais para que possam se engajar em um tipo de atividade que não dominam. Esses usuários representam uma população diversificada em termos de conhecimento, habilidades, interesses e estilos de aprendizagem. Além disso, são usuários que vão desenvolver habilidades e compreensão por meio da

interação com o software, portanto, suas necessidades de apoio e funcionalidades vão mudar com o passar do tempo.

3.2.1 Recomendações para o desenho centrado no aprendiz

Devido às peculiaridades dos aprendizes, o desenho de interfaces para esse tipo de audiência deve ser guiado por outros métodos e princípios não contemplados pelo Desenho Centrado no Usuário. Bruckman, Bandlow e Forte (2002) destacam as seguintes atividades envolvidas no processo do Desenho Centrado no Aprendiz:

• Análisar as necessidades

- Dos aprendizes
- Dos professores

• Selecionar uma abordagem pedagógica

• Selecionar um tipo de mídia/tecnologia

• Prototipagem

- Principal aplicação
- Apoio curricular
- Estratégias de avaliação do aprendizado

• Avaliar a interface

- Usabilidade
- Resultados de aprendizagem

• Desenho do sistema

• Avaliação final

- Usabilidade
- Resultados de aprendizagem

Para softwares e outros sistemas interativos que serão usados em ambiente escolar, é necessário entender não apenas os estudantes como também os professores. Após identificar as necessidades da audiência, é preciso selecionar uma abordagem pedagógica que oriente o método de ensino e aprendizagem. Em seguida surge a seleção do tipo de mídia (aplicativo para smartphone, para tablet, website, jogo, etc.). Na fase de construção do protótipo, é preciso considerar não apenas questões tecnológicas relacionadas à aplicação como também funções que ofereçam o apoio curricular e avaliação do desenvolvimento do estudante.

Antes de passar para a implementação, é fundamental fazer uma avaliação formativa para saber que componentes da interface podem ser melhorados tendo em vista a usabilidade e os objetivos de aprendizagem. Após a avaliação formativa e implementação do sistema interativo, cabe uma avaliação final (somativa). "A avaliação somativa final serve para documentar a efetividade do design e justificar seu uso pelos aprendizes e professores. A avaliação somativa deve prestar atenção de forma semelhante à usabilidade e aos objetivos de aprendizagem" (Bruckman et al., 2002, p. 803). A avaliação dos resultados de aprendizagem pode ser obtida por métodos quantitativos ou qualitativos coerentes com a abordagem pedagógica escolhida.

É importante destacar que o processo descrito acima não é rígido e nem sempre é possível ser aplicado na mesma sequência. De todo modo, nota-se que as etapas do Desenho de Centrado no Aprendiz tendem a ser mais demoradas que no Desenho Centrado no Usuário. Sobre essa questão, Brucknam et al. (2002) acrescentam:

> Enquanto em alguns casos é possível coletar dados válidos de usabilidade em uma simples sessão, a aprendizagem tipicamente ocorre por longos períodos. Para conseguir dados significativos, a maioria dos estudos em sala de aula podem ocorrer ao longo de semanas ou meses. Além disso, as pesquisas em sala de aula precisam se encaixar em um período apropriado do ano escolar. (..) É frequente levar alguns anos para completar o processo do desenho centrado no usuário. Na comunidade científica, uma equipe pode estudar e desenvolver uma parte da tecnologia educativa por muitos anos. No ambiente comercial, produtos educativos precisam estar no mercado rapidamente, e este processo formal de design é raramente usado. (p. 803, tradução livre)

Mesmo que não seja possível seguir todo o processo idealizado por pesquisadores da área do Desenho Centrado no Aprendiz, há recomendações que não podem ser ignoradas. Descatacamos as 11 heurísticas do desenho de interface para aprendizagem propostas por Dorian Peters (2013) a partir da adaptação das heurísticas de Nielsen. São recomendações úteis porque ajudam no desenho e avaliação de interfaces que priorizam a travessia do golfo de expertise sem descartar a usabilidade.

1. Relevância dos elementos multimedia e redução da carga cognitiva extrínseca: As causas de sobrecarga cognitiva como imagens, detalhes decorativos e visuais e outros elementos de media que não estão diretamente relacionados com o objetivo de aprendizagem ou requerimentos da interação devem ser evitadas.

2. Controle e liberdade do aprendiz: O nível de controle por parte de quem aprende, endossado pelo design de navegação, arquitetura e interação, deve ser apropriado às características da audiência e à abordagem pedagógica.

3. Suporte aos objetivos de aprendizagem: A interface gráfica, os conteúdos gráficos e o design de interação devem apoiar os objetivos de aprendizagem tal como definidos pelos designers instrucionais ou educadores.

4. Alinhamento com necessidades específicas do aprendiz: O design deve ser influenciado por características específicas da audiência, tais como: conhecimento prévio, cultura, literacia, literacia computacional, literacia visual, idade, cultura profissional e quaisquer outros aspetos que possam informar as decisões de design.

5. Adequabilidade visual e emocional (*look and feel*): Os componentes do desenho da interface devem refletir uma imagem adequada da audiência, mensagem e conteúdo da experiência de aprendizagem (por exemplo: nem demasiado infantilizado ou demasiado sério ao ponto de impossibilitar o interesse por parte de crianças, ou insensível ao ponto de se tornar ofensivo quando em contextos de assuntos sérios).

6. Suporte aos aspetos cognitivos de aprendizagem: O design deve apoiar os aspetos cognitivos de aprendizagem que sejam relevantes para a experiência (construção racional, carga cognitiva, resolução de problemas, interação social, etc.) tais como definidos por uma ou mais teorias da aprendizagem. Os obstáculos aos aspectos cognitivos da aprendizagem devem ser considerados como erros no design de interfaces de aprendizagem.

7. Suporte aos aspectos afetivos de aprendizagem: O design deve apoiar os aspetos afetivos da aprendizagem que são relevantes para os objetivos pedagógicos (dentro dos limites baseados em evidências científicas). Os obstáculos aos

aspectos afectivos da aprendizagem devem considerados como erros no contexto do design de interfaces de aprendizagem.

8. Media e ferramentas apropriados: Utilização de meios, dispositivos e ferramentas que são apropriados ao tipo de aprendizagem ou atividade.
9. Acessibilidade: Se é acessível a todos os indivíduos em processo de aprendizagem, independentemente da existência de algum tipo de incapacidade do aprendiz, do tipo de dispositivo utilizado ou do nível de literacia tecnológica.
10. Usabilidade: Conformidade em relação às recomendações de usabilidade e às melhores práticas.
11. Feedback e capacidade de resposta: O design deve permitir a provisão de feedback operacional e instrucional. O feedback deve ser intrínseco sempre que possível e, quando extrínseco, deve ser colocado de forma próxima ao item relevante e deixar espaço para respostas densas do ponto de vista das instruções. O feedback operacional deve ser providenciado de modo instantâneo.

3.3 Recomendações para o desenho de interfaces para crianças

As *guidelines* propostas por pesquisadores interessados no Desenho Centrado no Aprendiz são direcionadas a uma população de usuários de qualquer faixa etária que tem em comum a necessidade de aprender um determinando assunto ou desenvolver algum tipo de habilidade por meio da sistema interativo. Para o desenho de interface de aprendizagem no contexto no ensino básico, devemos considerar também as habilidades e interesses das crianças diante dos sistemas interativos.

Os estudos sobre Interação Criança-Computador indicam boas práticas de desenho, avaliação e implementação de sistemas computacionais interativos para o público infantil. Hourcade (2015) define Interação Criança-Computador como um campo que estuda como desenhar tecnologias interativas para crianças e como elas podem, a partir dessa interação, ter experiências com efeitos positivos para o seu desenvolvimento.

O autor também afirma que os designers devem conheçer as pesquisas sobre desenvolvimento infantil para desenhar tecnologias apropriadas para diferentes fases da

infância. Esta visão também é compartilhada por outros pesquisadores que defendem que a tecnologia deve ser desenhada a partir do que já é conhecido sobre o desenvolvimento das crianças (Brukman et al., 2002).

Entre as teorias do desenvolvimento cognitivo que tiveram maior impacto nos estudos sobre interação criança computador destacam-se o construtivismo de Piaget, construcionismo de Papert e abordagens socioculturais inspiradas nos estudos de Vygotsky (Hourcade, 2015. p. 7).

Como vimos no capítulo 2, o construtivismo observa a criança como ativa construtora de conhecimento por meio de suas experiências. Já Papert expande a ideia de Piaget com sua proposta constricionista, segundo a qual as crianças aprendem melhor quando estão conscientemente engajadas na construção de artefatos que podem ser compartilhados. Por sua vez, Vygotsky é um dos pioneios a ressaltar a importância da interação social no desenvolvimento das crianças.

Hourcade identificou, a partir da sistematização dos estudos na área, os 10 pilares da Interação Criança-Computador:

1. Trabalho com equipes interdisciplinares – Atualmente, a maior parte dos sistemas interativos para crianças resulta do trabalho de equipes. Os melhores projetos envolvem profissionais de diferentes áreas, com experiência em métodos de design e avaliação de interfaces, desenvolvimento de softwares, especialistas na compreensão de um público específico relacionado à infância, como crianças, pais, professores, educadores e psicólogos, além de especialistas no domínio de atividade abrangido pela tecnologia.

2. Profundo engajamento dos *stakeholders* – O envolvimento de stakeholders nas diversas etapas do processo do design aumenta a probabilidade de sucesso do projeto. Também é importante envolver as crianças no processo para facilitar uma abordagem de desenho centrada em suas necessidades. É importante ressaltar que as crianças não são as únicas afetadas pelo uso da tecnologia, os adultos com os quais elas interagem também têm um importante papel.

3. Avaliação contínua – Para entender o real impacto do uso da tecnologia por crianças, é necessário avaliar por um longo período, em diversas fases do projeto.

4. Desenho de uma ecologia, não apenas de uma tecnologia – A abordagem ecológica leva em consideração o que está acontecendo no ambiente. Como esta abordagem pode afetar o processo do design? Os designers devem considerar o amplo contexto de uso da tecnologia. Por isso, recomenda-se observar os espaços físicos onde a tecnologia será usada e as pessoas que poderão estar presentes no mesmo ambiente das crianças durante a interação com a interface (pais, irmãos, colegas, professores, etc).

5. Tornar prático para a realidade das crianças – A tecnologia desenhada para criança deve funcionar no contexto desse público específico.

6. Personalização – As crianças têm necessidades e interesses diversos, por isso a tecnologia que permite a personalização terá vantagens.

7. Ter atenção à hierarquia de habilidades – As equipes de designers precisam ter atenção ao conjunto de habilidades necessário para o uso de um sistema interativo e certificar-se de que as crianças que usarão a tecnologia possuem essas habilidades.

8. Dar suporte à criatividade – tecnologias interativas devem dar suporte a um amplo leque de atividades criativas, tais como storyteling, autoria musical, animações, entre outros. O objetivo é permitir que a criança sinta-se motivada na medida em que se envolve em experiências significativas por meio da criação ou da construção.

9. Ampliar a conexão humana – O campo da interação criança computador confere bastante atenção a tecnologias que permitem a comunicação e a colaboração. Isso porque a interação social é fundamental para o desenvolvimento infantil. Há duas abordagens que facilitam a comunicação e a colaboração: frente a frente, com foco nas pessoas que estão fisicamente próximas; e remota, com pessoas que estão separadas geograficamente.

10. Permitir diferentes experiências - Os sistemas interativos que ampliam experiências físicas seja em ambientes abertos ou fechados oferecem um desenvolvimento positivo para as crianças.

3.4 Recomendações para interfaces de introdução ao pensamento computacional

Uma característica comum entre as ferramentas mais usadas para a introdução ao pensamento computacional é o baixo limiar de entrada para reduzir qualquer barreira à interação de estudantes completamente novatos no domínio de atividade. Dessa forma, o estudante pode realizar uma atividade inicial rapidamente.

Além disso, as ferramentas também dão suporte à interação de usuários mais experientes, acompanhando a evolução dos estudantes. Tais características vão ao encontro das recomendações de design de softwares educativos propostas por Papert (1980), para quem as interfaces deveriam ter "piso baixo" e "teto alto" (*low floor/high ceiling*), fáceis de usar e viáveis para a criação de projetos mais complexos.

Repenning, Webb e Ioannidou (2010) propõem uma *checklist* para o desenho de interfaces de introdução ao pensamento computacional. Para os autores, além de "piso baixo" e "teto alto", as ferramentas devem:

1. Possuir andaimes (*scaffolds*): o currículo fornece uma sequência de passos com a gestão de habilidades e desafios para acompanhar a ferramenta;

2. Permitir transferência: a ferramenta deve facilitar a transição para outras linguagens de programação e a aplicação dos conceitos e práticas das Ciências da Computação em diferentes contextos.

3. Apoiar a equidade: as atividades devem ser acessíveis e motivacionais, sem fronteira de gêneros e etnias.

4. Ser sistêmica e sustentável: a combinação entre ferramenta e currículo pode ser usada por todos os professores para ensinar todos os estudantes (por exemplo: permitir o treinamento de professores, alinhamento de normas etc).

No capítulo 1, apresentamos exemplos de ferramentas que se encaixam nessas características. No próximo, são apresentadas as características da interface do Scratch e as estratégias adotadas pelos autores da plataforma para ajudar os aprendizes a desenvolver habilidades típicas do pensamento computacional.

Capítulo 4: Scratch

Scratch é uma das ferramentas mais utilizadas em escolas para o desenvolvimento do pensamento computacional. Neste capítulo, analisamos os princípios que guiaram o desenho da interface do Scratch. Iniciamos por uma descrição das características gerais da ferramenta e do contexto em que foi desenvolvida. Em seguida, apresentamos a teoria da aprendizagem que influenciou os seus desenvolvedores. Então identificamos, a partir da análise dos componentes da interface, as decisões de design que ajudam os usuários do programa a desenvolver o pensamento computacional.

4.1 O que é o Scratch?

O Scratch é uma linguagem de programação visual e uma comunidade online que possibilita a criação e o compartilhamento de projetos interativos, como animações, jogos, tutoriais, músicas e simulações (Maloney et al., 2010). O projeto foi desenvolvido por pesquisadores do Lifelong Kindergarten Group, do MIT Media Lab. Desde o seu lançamento, em 2007, o Scratch registra mais de 13 milhões de projetos e 10 milhões de usuários, a maioria na faixa etária de 10 a 12 anos. O programa é de acesso livre e está disponível em mais de 50 idiomas.

O Scratch foi inicialmente projetado em 2003 para desenvolver as competências digitais de crianças e adolescentes de 8 a 16 anos participantes do Computer Clubhouse, um programa comunitário extra-classe de aprendizagem de tecnologia (Proposal, 2003). Porém, devido às características de sua interface (intuitiva e usável), o suporte à criação de diferentes tipos de projetos e o apoio pedagógico oferecido por seus autores, o programa passou a ser usado por um público de diferentes faixas etárias e em diversos ambientes de aprendizagem, como escolas, museus e universidades.

O Scratch é usado, por exemplo, nas aulas iniciais de curso de Introdução às Ciências da Computação de Havard (*CS50: Introduction to Computer Science*). Em Portugal, há diversas iniciativas que promovem o seu uso em ambiente escolar, entre as quais destacam-se o EduScratch e o projeto Iniciação à Programação no 1.º Ciclo do Ensino Básico, promovidos pela Direcção Geral de Educação (DGE) por meio da Equipa de Recursos e Tecnologias Educativas (ERTE).

O principal objetivo do Scratch é introduzir crianças e jovens ao mundo da programação para que eles se tornem fluentes digitais. Para Resnick et al. (2009), ser digitalmente fluente envolve não apenas saber como usar as ferramentas tecnológicas, mas também saber como construir coisas significativas com estas ferramentas. O autor também afirma que a programação no ambiente do Scratch ajuda a desenvolver aptidões necessárias para o pensamento computacional.

> A fluência digital não requer apenas a capacidade de conversar, navegar e interagir, mas também a de projetar, criar e inventar com as novas mídias. (..) Para fazer isso, é necessário aprender algum tipo de linguagem de programação. A habilidade de programar gera importantes benefícios. Por exemplo, expande consideravelmente a gama do que você pode criar (e como você pode se expressar) com o computador. Também amplia a gama do que você pode aprender. Em particular, a programação é um suporte ao "pensamento computacional", ajudando você a aprender importantes estratégias de resolução de problemas e de design (como modularização e desenho de interação) que podem ser transportadas para outros domínios que não exigem programação. E uma vez que a programação envolve a criação de representações externas de processos de resolução de problemas, programar fornece-lhe a oportunidade de refletir sobre o seu próprio pensamento. (p. 62, tradução livre)

Para atender a necessidade de desenvolver um ambiente de programação fácil e interessante para crianças e jovens, a equipe do Scratch optou pela linguagem de programação visual baseada em blocos de construção (*building-block programming)*, que podem ser encaixados uns aos outros como as peças de Lego para criar programas (Figura 4.1).

Figura 4.1: Exemplo de *script* construído com blocos de programação do Scratch

O formato de cada bloco gráfico indica como ele pode ser conectado a outro. Diferente das tradicionais linguagens de programação baseadas em texto, a gramática do Scratch é visual. Sua sintaxe é indicada pelo formato dos blocos e conectores que só podem ser encaixados se a combinação estiver sintaticamente correta (Maloney et al., 2004; Resnick & Rosenbaum, 2013).

Maloney et al. (2004) apontam outras características fundamentais do Scratch:

- Manipulação de mídia: permite montar programas que controlam e misturam imagens, gráficos, músicas e som;
- Compartilhamento e colaboração: permite divulgar os projetos realizados assim como reutilizar e adaptar projetos criados por outras pessoas;
- Integração com o mundo físico: permite que as criações possam responder a diferentes estímulos do mundo físico por meio da conexão do programa a sensores externos, como os sensores de som e detectores de movimento;
- Tradução em diversos idiomas: permite que o usuário possa escolher entre diversas línguas para que o programa tenha alcance internacional.

O ambiente de edição de projetos no Scratch foi construído em Squeak, que é uma linguagem de programação orientada a objetos derivada da linguagem Smalltalk-80.

4.2 Base pedagógica

O Scratch está alinhado a uma visão construcionista da aprendizagem. A equipe de pesquisadores do Lifelong Kindergarten Group, responsável pelo programa, dá continuidade a uma tradição de projetos iniciada no desenvolvimento da linguagem de programação Logo, pela equipe de Seymour Papert, no fim dos anos 60. Dasgupta e Resnick (2014) explicam como ocorre essa influência:

> O núcleo da abordagem de design do Scratch baseia-se nos princípios do construcionismo, já que o Scratch permite aos aprendizes construir artefatos públicos (projetos). Esses projetos se tornam "objetos com os quais pensar", em que o processo de construção oferece oportunidades para envolver-se não apenas no pensamento computacional, mas também no processo de "pensar sobre o pensamento". (..) Tudo isso está situado no contexto da comunidade on-line do Scratch, um espaço para troca de idéias, aprendizagem e colaboração com os outros. (p. 73)

O principal desafio para o desenho de interfaces de aprendizagem apoiado na teoria construcionista é o desenvolvimento de um espaço virtual que envolva os aprendizes em processos de criação, construção, experimentação e compartilhamento. Essas atividades estão na base dos quatro princípios que orientaram o desenho do Scratch: projetos, parceria, paixão e experimentação (Resnick, 2014). Vejamos como esses princípios podem ser identificados na interface do programa:

• Projetos: para os construcionistas, as pessoas aprendem melhor quando estão envolvidas em projetos com as quais se identificam. O Scratch foi desenhado com a ideia de projetos em mente e, por isso, oferece uma variedade de ferramentas que permitem a construção de diferentes tipos de artefatos digitais (Resnick, 2014).

• Parceria: as pessoas também aprendem melhor quando compartilham suas ideias e colaboram na construção de projetos. A valorização da dimensão social da aprendizagem foi determinante para duas importantes decisões do design do Scratch: a criação de uma comunidade online para o compartilhamento dos programas e a possibilidade de remistura (*remix*) dos projetos feitos por outros usuários.

• Paixão: a teoria construcionista sugere que as pessoas também aprendem melhor quando se envolvem em projetos que tenham um significado pessoal emocial. Se estiverem motivadas, elas podem superar com mais facilidade as dificuldades relacionadas ao projeto e ter persistência diante dos desafios. Diante da variedade de interesses dos aprendizes, a equipe do Scratch optou por permitir o uso de diferentes tipos de mídia, como músicas, sons, fotografias e gráficos (Resnick, 2014). A variedade de recursos disponíveis explica a diversidade de projetos criados, como jogos, tutoriais, animações, entre outros (Figura 4.2).

• Experimentação: Scratch foi desenhado para estimular a experimentação livre, sem a necessidade de planejamento prévio, atitudes que estão no cerne do conceito de "*tinkerability*", palavra sem tradução satisfatória em português.

Os quatro princípios convergem para a criação de um ambiente digital que estimula a concepção e desenvolvimento de projetos para a promoção da aprendizagem. Dessa forma, o Scratch (cujo slogan é "imagina, programa, partilha") representa uma ferramenta para a concretização de uma ideia-chave da teoria construcionista: aprendizagem por meio da criação de artefatos (*learning by design*).

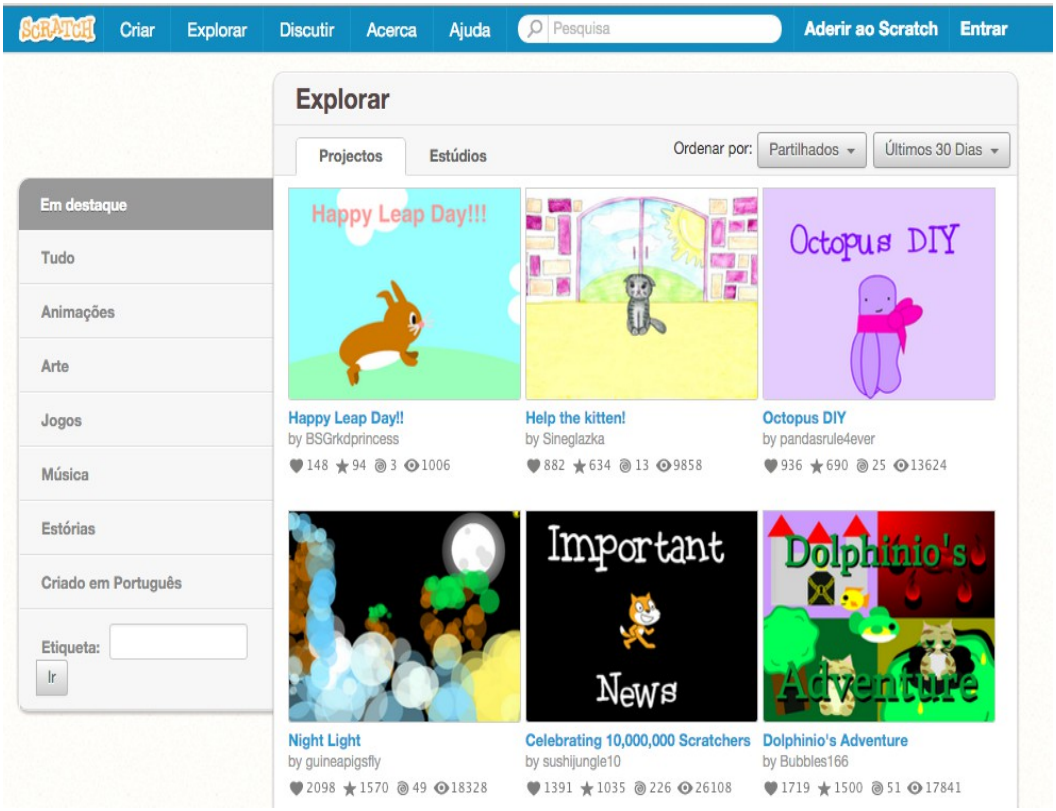


Figura 4.2: Mural de exposição dos projetos

4.3 Pensamento computacional no Scratch

Os autores do programa creem que a habilidade de programar ajuda a pensar computacionalmente. Resnick compara o pensamento computacional à literacia e a programação à escrita para defender a importância do domínio das duas habilidades (NRC, 2010, p. 13).

Brennam e Resnick (2012) desenvolveram um quadro referencial composto por conceitos (Quadro 4.1), práticas (Quadro 4.2) e perspectivas (Quadro 4.3) do pensamento computacional no Scratch. Estas três dimensões tem sido utilizadas como referência para avaliação do desenvolvimento do pensamento computacional em diversos contextos educativos que utilizam o Scratch (Ramos & Espadeiro, 2014).

Para apoiar esta avaliação, os autores também publicaram um guia curricular intitulado *Creative Computing*, disponível em várias línguas com recomendações para análise dos projetos. A comunidade online ScratchEd (http://scratched.gse.harvard.edu/) concentra a maioria dos recursos de apoio pedagógico que ampliam as potencialidades de uso da ferramenta.

Quadro 4.1: Conceitos computacionais

Conceito	**Descrição**
Sequência	Identificar uma série de etapas de uma tarefa
Ciclos (loops)	Executar a mesma sequência várias vezes
Execução em paralelo	Fazer as ações decorrerem ao mesmo tempo
Eventos	Fazer um acontecimento causar outro acontecimento
Condições	Tomar decisões com base em condições
Operadores	Expressar operações matemáticas e lógicas
Dados	Armazenar, recuperar e atualizar valores

Quadro 4.2: Práticas computacionais

Prática	**Descrição**
Ação iterativa e incremental	Desenvolver, verificar se funciona e, em seguida, continuar a desenvolver
Teste e depuração	Certificar-se de que tudo funciona e encontrar e corrigir erros
Reutilização e reformulação	Fazer algo utilizando o que já foi feito em outros projetos
Abstração e modulação	Construir algo grande unindo conjuntos de partes mais pequenas

Quadro 4.3: Perspectivas computacionais

Perspectiva	**Descrição**
Expressar	Perceber que a computação é um meio de criação. "Eu posso criar."
Conectar	Reconhecer a vantagem de criar com e para outros. "Eu posso ter novas ideias quando tenho acesso a outros."
Questionar	Sentir que se pode fazer perguntas sobre o mundo. "Eu posso (utilizar a computação para) suscitar questões que façam sentido (com entes computacionais) para o mundo.”

Na próxima seção, identificamos os principais componentes do ambiente de criação de projetos e da linguagem de programação do Scratch que dão suporte ao desenvolvimento das três dimensões do pensamento computacional definidas pelos autores do programa.

4.5 Interface do ambiente de programação

A equipe responsável pelo desenho da interface do Scratch optou por exibir todos os recursos de edição em uma única página dividida em 3 painéis e menu de navegação (Figura 4.3).

A coluna da esquerda está dividida em duas partes: o plano superior, formado por *Stage* (Palco) e *Sprites* (objetos animados), permite visualizar o resultado de cada instrução; A *Sprite List* (lista de Sprites), no plano inferior, permite ao usuário selecionar diferentes palcos e objetos. À direita está a aba *Scripts* (guiões), composta pelas paletas dos blocos visuais de programação e pela *Scripts Area* (área de scripts). A coluna da direita também contém as abas *Costumes* (Fantasias) e *Sounds* (Sons), que permitem a customização dos *Sprites* e incorporação de conteúdo multimídia.

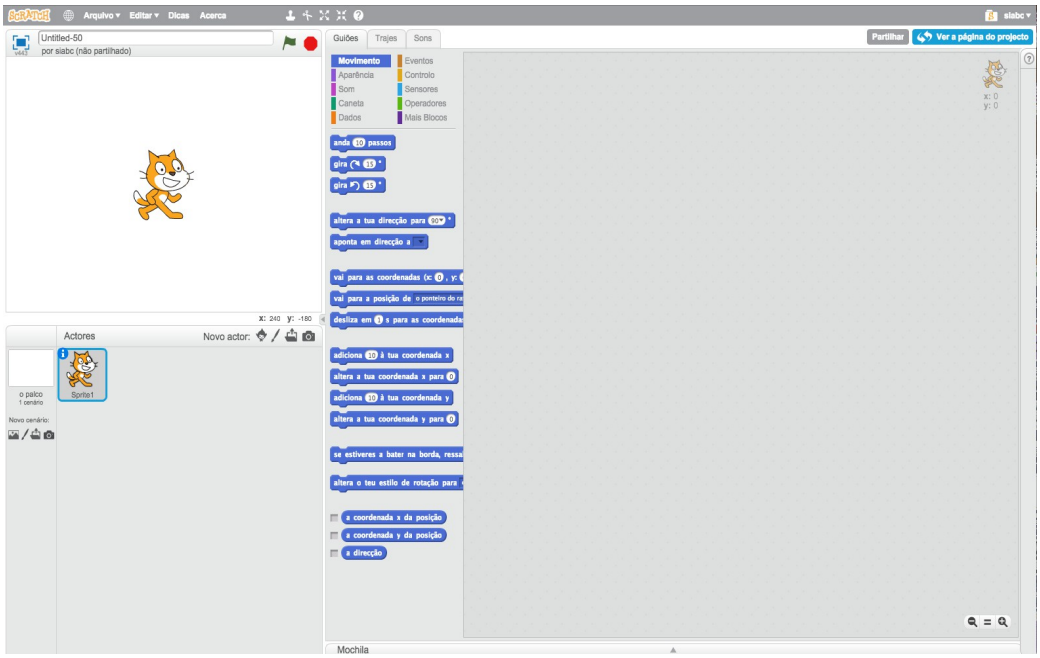
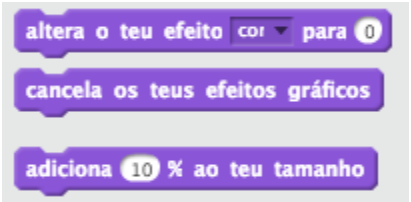
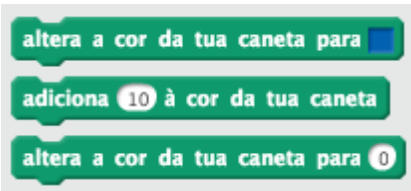


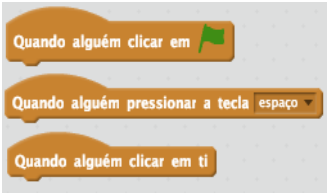
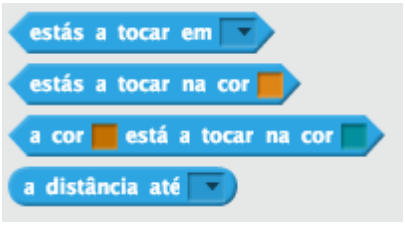
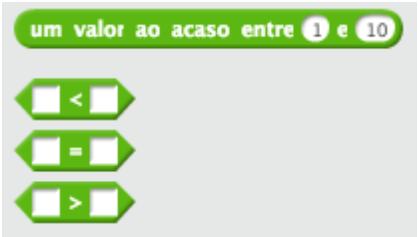
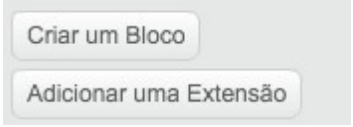
Figura 4.3: Interface do ambiente de programação do Scratch

Para criar os programas, o usuário deve selecionar um bloco gráfico da paleta central, arrastá-lo com o rato para a área de script e conectá-lo a outros blocos. Não é necessário digitar comandos como em linguagens de programação baseadas em texto. Os blocos estão sempre visíveis para facilitar e estimular a criação dos *scripts* (Maloney et al., 2004; 2010).

Os blocos são diferenciados por cores e organizados em 10 categorias. Descrevemos cada categoria no Quadro 4.4:

Quadro 4.4: Categorias dos blocos de comando do Scratch

Categoria	Descrição	Exemplos
Movimento	Controlam movimentos dos Sprites	
Aparência	Cotrolam a aparência dos Sprites e Stages	
Som	Controlam a reprodução de sons	
Caneta	Controlam o desenho que pode ser feito no Stage	

Dados	Permitem criar variáveis e listas	
Eventos	Permitem alterar a execução de um Script conforme determidado evento	
Controle	Controlam o fluxo de processamento dos Scripts	
Sensores	Detectam eventos, como contato e distância, relacionados aos Sprites	
Operadores	Executam operações booleanas, manipulação de strings e funções matemáticas	
Mais Blocos	Blocos criados pelo usuários	

Scratch segue uma linha minimalista para simplificar a realização dos projetos. Os blocos, por exemplo, foram desenhados cuidadosamente para não ter um grande número de parâmetros. Para a equipe, é importante minimizar o número de blocos de comandos para que não ocupe muito espaço na paleta de blocos. Evita-se que o usuário utilize a barra de rolagem da tela para encontrar as soluções para os seus projetos.

Mesmo com número limitados de blocos de comandos, a linguagem de programação do Scratch permite explorar as três dimensões do pensamento computacional referidas na seção anterior. Vejamos de que forma o uso dos blocos ajuda o aprendiz a entender os conceitos computacionais definidos por Brennam e Resnick (2012):

a) Sequência: ao criar programas em Scratch, os utilizadores precisam organizar as instruções de forma sistemática (Figura 4.4), desenvolvendo a habilidade do pensamento algorítmico. A área de script oferece *feedback* visual de cada comando em execução. Este recurso permite ao usuário entender o funcionamento da sequência de algoritmos criada.

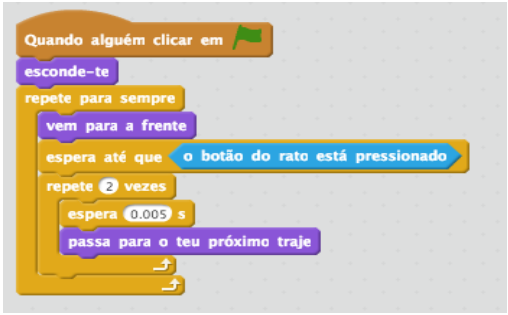


Figura 4.4: Exemplo de blocos organizados de forma sistemática

b) Ciclos ou iteração (*loops*): os blocos da categoria Controle (Figura 4.5) podem ser usados para entender o conceito de ciclos, já que permitem executar a mesma sequência de funções várias vezes.

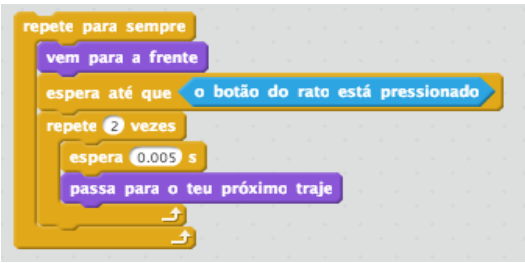


Figura 4.5: Exemplo de blocos de controle dos ciclos

c) Execução em paralelo: é possível executar dois conjuntos de instruções ao mesmo tempo (Figura 4.6).

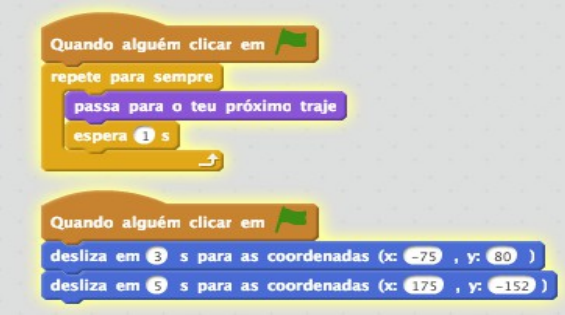


Figura 4.6: Exemplo de execução em paralelo

d) Evento: ao utilizar os blocos "Quando alguém clicar na bandeira verde" ou "Quando alguém pressionar a tecla _ " (Figura 4.7), o usuário aprende como fazer um acontecimento causar outro acontecimento.

Figura 4.7: Exemplo de script que indica evento

e) Condições: blocos "se" e "senão" (Figura 4.8) permitem trabalhar com instruções condicionais.

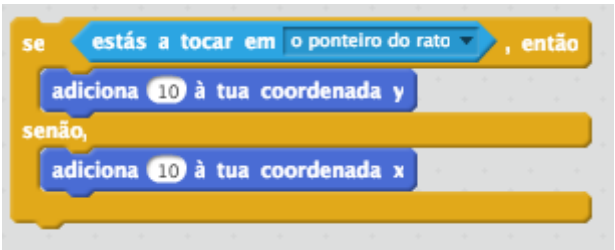


Figura 4.8: Exemplo de blocos de instruções condicionais

f) Operadores: os usuários podem realizar operações matemáticas e lógicas por meio do conjunto de blocos da categoria Operadores (Figura 4.9). Exemplos de blocos de lógica booleana:

Figura 4.9: Exemplo de blocos com operadores booleanos

g) Manipulação dados: o Scratch permite que os usuários façam operações com dados por meio da manipulação de variáveis e listas. Para diminuir a complexidade deste tipo de tarefa, os autores optaram por adicionar um monitor (Figura 4.10) que aparece no Palco assim que uma variável ou lista é criada.

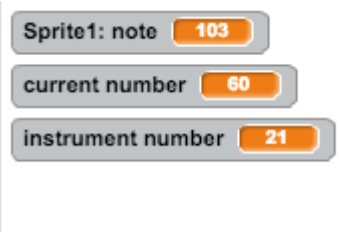


Figura 4.10: Monitor de variáveis

Monitorar os efeitos de cada mudança de variáries facilita o entendimento desse conceito em programação. Segundo Maloney et al. (2010), nas primeiras versões do Scratch, uma variável recém-criada não era automaticamente exibida na área do palco (*stage*) e os comandos necessários para fazer com que isso acontecesse não eram óbvios. Por isso, muitos usuários deixavam de usar as variáveis porque não percebiam o seu funcionamento por meio da interface. Após a mudança do design, com a implementação do monitor de variáveis, muitos usuários começaram a usar este recurso.

Observamos que o ambiente de programação do Scratch também tem funcionalidades que permitem explorar diversas práticas comuns aos cientistas da computação, como iteração, teste, depuração, reutilização e reformulação de projetos.

O usuário pode mudar a disposição dos blocos mesmo com o programa em execução ou executar um bloco isoladamente e conferir seus efeitos. Estas funcionalidades estimulam a experimentação sem planejamento prévio (*tinkerability*). Além disso, o usuário pode perceber o conceito de execução em paralelo, já que é possível executar diversas instruções ao mesmo tempo por diferentes conjuntos de blocos.

4.6 Considerações sobre a ferramenta

A análise do Scratch indica que os seus autores adotam a metodologia de Desenho Centrado no Aprendiz para ajudar crianças e jovens a aprender a programar e, por conseguinte, adquirir fluência digital e desenvolver o pensamento computacional. Nesta perspectiva, a linguagem de programação visual serve como andaime (*scaffolding*) para apoiar os aprendizes na travessia do golfo de aprendizagem. Resnick (2013) enfatiza que o objetivo não é apenas aprender a programar, mas programar para aprender.

Observamos que cada componente da interface do programa pode ser explicado por referências do design, do construcionismo e das Ciências da Computação. A combinação dessas fontes e o profundo conhecimento da audiência por meio de estudos empíricos resultaram em uma plataforma de construção de projetos que oferece as ferramentas necessárias para a criação, a experimentação e o compartilhamento de artefatos digitais.

O ambiente de programação do Scratch pode ser comparado a uma caixa de ferramenta que está sempre aberta para que os aprendizes identifiquem rapidamente os recursos disponíveis. Dessa forma, os autores evitam a carga cognitiva extrínseca ao objetivo de aprendizagem e permitem que os aprendizes concentrem-se na construção de projetos.

Segundo Brenann, Balch e Chung (2014), à medida que avançam na criação, os usuários passam a ter uma nova visão de si mesmos e da tecnologia. Essa mudança de perspectiva, consiste em reconhecer a capacidade de se expressar, de se conectar e de questionar por meio dos sistemas computacionais.

> O engajamento na criação de artefatos computacionais prepara os jovens para algo mais do que carreiras como cientistas de computação ou como programadores e apoia o seu desenvolvimento como pensadores computacionais – indivíduos que podem recorrer a conceitos computacionais, práticas e perspetivas em todos os aspetos das suas vidas, em várias disciplinas e contextos. (p. 1, tradução livre)

É importante ressaltar que os resultados da aprendizagem também vão depender da forma como a ferramenta é utilizada no ambiente educacional. A atenção a esse aspecto tem motivado a realização de diversas pesquisas direcionadas para a avaliação pedagógica decorrente do uso do Scratch (Marques, 2009; Sousa & Lencastre, 2014; Ramos & Espadeiro, 2014; López, González & Cano, 2016). Os estudos corroboram a escolha da ferramenta como uma opção válida para introdução ao pensamento computacional no ensino básico.

A avaliação positiva da comunidade acadêmica e o engajamento dos aprendizes, confirmado pelo número de projetos criados e compartilhados (13 milhões), indicam que os autores do Scratch estão alcançando os objetivos educativos, lúdicos e sociais propostos pela ferramenta. Para os designers, o olhar atento ao programa também é uma oportunidade de aprendizagem.

Conclusão

Há um movimento crescente de educadores e entidades representativas das ciências da computação que aceitaram a proposta desafiadora de Jeannette Wing (2006) de promover o pensamento computacional desde os primeiros anos de escolaridade. Acreditamos que essa disseminação pode ser facilitada pela exploração de ambientes computacionais de apoio à aprendizagem que estimulem práticas e atitudes inerentes ao pensamento computacional, tais como: formular problemas de forma que permita usar um computador e outras ferramentas para solucioná-los; organizar e analisar dados de forma lógica; representar dados através de abstrações, como modelos e simulações; automatizar soluções através de pensamento algorítmico; identificar, analisar e implementar soluções possíveis com o objetivo de alcançar a mais eficiente combinação de passos e recursos; generalizar e transferir esse processo de resolução de problemas a uma grande variedade de um mesmo tipo de problema (CSTA, 2012).

O fio condutor desta dissertação foi a busca por recomendações que indicassem o melhor caminho para o desenho de interfaces para a aprendizagem do pensamento computacional desde o ensino básico. Reunimos relevantes estudos provenientes do campo da Interação Homem-Computador que forneceram orientações para o desenho, implementação e avaliação de interfaces de aprendizagem apropriadas à idade e ao desenvolvimento cognitivo da audiência visada. Demonstramos, por meio da análise da ferramenta Scratch, o resultado concreto de decisões de design guiadas por esses princípios.

Partimos do pressuposto de que a abordagem de Desenho Centrado no Usuário, embora útil, não era suficiente para orientar o desenho de interfaces de aprendizagem. A pesquisa demonstrou que o ponto de partida deve ser o Desenho Centrado no Aprendiz, por um motivo: o aprendiz é diferente do usuário. Enquanto o principal desafio da primeira abordagem é ajudar o usuário a atravessar os golfos de execução e avaliação durante a interação com o sistema interativo, a segunda concentra-se em ajudar o aprendiz a superar um terceiro golfo: o de aprendizagem.

A partir desse quadro de referência, consideramos que o desenho de interfaces para aprendizagem do pensamento computacional no ensino básico deve ser conduzido por equipes multidisciplinares que tenham conhecimento sobre os seguintes aspectos:

a) Pensamento computacional

Os designers precisam ter compreensão de diferentes aspectos do contexto de aprendizagem, o que inclui os tipos de atividades, vocabulário e informações factuais relacionados ao domínio de estudo. No que se refere ao pensamento computacional, recomendamos que as equipes conheçam os objetivos de aprendizagem definidos por programas curriculares que incluem pensamento computacional no rol de competências digitais desenvolvidas no Ensino Básico. O capítulo 1 desta dissertação abordou esses aspectos e apresenta importantes fontes de informação.

b) Fundamentos da aprendizagem

É importante saber como as pessoas aprendem. Apresentamos, no capítulo 2, os fundamentos das principais teorias da aprendizagem. Também explicamos como o behaviorismo, cognitivismo, construtivismo e construcionismo podem orientar decisões de design de componentes da interface.

c) Heurísticas de usabilidade para interfaces de aprendizagem

Os designers devem priorizar a travessia do golfo de aprendizagem sem descartar o papel e o lugar da usabilidade de aplicações interativas. Essa tarefa pode ser baseada nas 11 heurísticas do desenho de interface para aprendizagem propostas por Dorian Peters a partir da adaptação das heurísticas de Nielsen:

- Relevância dos elementos multimedia e redução da carga cognitiva extrínseca;
- Controle e liberdade do aprendiz;
- Suporte aos objetivos de aprendizagem;
- Alinhamento com necessidades específicas do aprendiz;
- Adequabilidade visual e emocional (look and feel);
- Suporte aos aspetos cognitivos de aprendizagem;
- Suporte aos aspectos afetivos de aprendizagem;
- Escolha de media e ferramentas apropriados;
- Acessibilidade;
- Usabilidade;
- Feedback e capacidade de resposta.

d) Interação criança-computador

Os ambientes computacionais usados no Ensino Básico têm como principais usuários um público predominantemente infantil. A interface deve assim ser apropriada à idade e ao desenvolvimento cognitivo da audiência e, por isso, os designers devem considerar também as habilidades e interesses das crianças diante dos sistemas interativos. Os estudos sobre Interação Criança-Computador fornecem os dez pilares que orientam as melhores práticas para o desenho de interfaces para esse público:

- Trabalho com equipes interdisciplinares
- Profundo engajamento dos *stakeholders*
- Avaliação contínua
- Desenho de uma ecologia, não apenas de uma tecnologia
- Tornar prático para a realidade das crianças
- Personalização
- Ter atenção à hierarquia de habilidades
- Dar suporte à criatividade
- Ampliar a conexão humana
- Permitir diferentes experiências

A análise efetuada nesta dissertação demonstrou que as características da interface do Scratch estão alinhadas com os três campos dos saberes que os designers precisam dominar para o desenho de interfaces para o pensamento computacional: teorias da aprendizagem, Interação Homem-Computador e Ciências da Computação. A ferramenta permite a aprendizagem de práticas, conceitos e perspectivas do pensamento computacional, está alinhada com a teoria construcionista da aprendizagem e adota uma linguagem de programação visual para estimular a participação de crianças a partir de 8 anos de idade.

Em suma, consideramos que o desenho de interface para o desenvolvimento do pensamento computacional no ensino básico pode ser guiado pelo conjunto de recomendações apresentado ao longo da dissertação. As equipes de designers têm um importante papel na disseminação do pensamento computacional e podem facilitar na

travessia do golfo de aprendizagem de competências digitais necessárias para o cidadão do século XXI.

Esperamos que a dissertação estimule novas pesquisas sobre outras dimensões do desenho de interfaces de aprendizagem que não foram aprofundados neste trabalho. É importante que os pesquisadores e profissionais também observem ferramentas de introdução ao pensamento computacional que não utilizam o recurso da programação. A análise de boas práticas pode estimular a criação de novos ambientes computacionais com outras estratégias de promoção da aprendizagem.

Referências bibliográficas

Ackermann, E. K. (2004). Constructing knowledge and transforming the world. in A learning zone of one's own: Sharing representations and flow in collaborative learning environments, 1, 15-37.

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal*, 5(7), 832 – 835.

A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities. Proposal to the National Science Foundation, 2003 (project funded 2003-2007). Co-PIs: Yasmin Kafai, John Maeda, John Maloney, Natalie Rusk.

Anderson, K. (2009). Guardian launches Open Platform tool to make online content available free. Disponível em: http://www.theguardian.com/media/2009/mar/10/guardian-open-platform

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48- 54.

Bates, A. (2015). Teaching in a Digital Age. Disponível em:

https://opentextbc.ca/teachinginadigitalage/

Bell, T. C. et al. (n.d.). *Computer Science Unplugged: Off-line activities and games for all ages*. [s.l: s.n.]. v. 23.

Benyon, D. (2013). Designing interactive systems: A comprehensive guide to HCI, UX and interaction design. (3rd ed) Trans-Atlantic Publications, Inc.

Blikstein, P., Martinez, S. L. & Pang, H. A. (ed.) (2015). Meaningful Making: Projects and Inspirations for FabLabs and Makerspaces. Disponível em:

http://fablearn.stanford.edu/fellows/sites/default/files/Blikstein_Martinez_Pang-Meaningful_Making_book.pdf

Blikstein, P. (2014). Seymour Papert's Legacy: Thinking About Learning, and Learning About Thinking. Disponível em: https://tltl.stanford.edu/content/seymour-papert-s-legacy-thinking-about-learning-and-learning-about-thinking

Bransford, J. D., Brown, A. L., & Cocking, R. R. (eds). 2000. How People Learn: Brain,

Mind, Experience and School. Washington D.C.: National Academy Press. Disponível em: http://www.nap.edu/catalog/9853/how-people-learn-brain-mind-experience-and-school-expanded-edition

Brennan, K., Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada.

Brenann, K., Balch, C & Chung, M. (2014) Creative Computing. An Introductory Computing Curriculum Using Scratch. Disponível em: http://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf

Bruckman, A., Bandlow, A. (2003). HCI for Kids. In: Jacko, J., Sears, A. (eds.) Human-Computer Interaction Handbook, NJ: Lawrence Erlbaum Associates, pp. 428–440.

Catlin, D., Woollard, J. (2014). Educational robots and computational thinking. In, Teaching Robotics & Teaching with Robotics (TRTWR) - Robotics in Education (RIE) 2014 Conference, Padova, Italy, Padova, Italy, Robotics in Education (RIE) 8pp.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C. & Woollard, J. (2015). *Computational thinking: A guide for teachers*. Disponível em: http://computingatschool.org.uk/computationalthinking

CSTA (2009). Computational Thinking Across the Curriculum. Disponível em: http://csta.acm.org/Curriculum/sub/CurrFiles/CTExamplesTable.pdf

CSTA (2011). Computational Thinking in K–12 Education leadership toolkit. Disponível em: http://csta.acm.org/Curriculum/sub/CurrFiles/471.11CTLeadershiptToolkit-SP-vF.pdf

CSTA. (2011). K–12 Computer Science Standards. Disponível em: https://csta.acm.org/Curriculum/sub/K12Standards.html

Dasgupta, S., & Resnick, M. (2014). Engaging Novices in Programming, Experimenting, and Learning with Data. ACM Inroads, vol. 5, no. 4, pp. 72-75.

Dix, A., Finlay, J., Abowd, G. D., & Beale, R. (2004). Human-Computer Interaction. Pearson-Prentice Hall.

Ertmer, P. A., Newby, T. J. (2013), Behaviorism, Cognitivism, Constructivism: Comparing Critical Features From an Instructional Design Perspective. Perf. Improvement Qrtly, 26: 43–71.

Ferrari, M. (2011). B. F. Skinner. Disponível em: http://educarparacrescer.abril.com.br/aprendizagem/bf-skinner-307060.shtml

Ferreira, Henrique da Costa. (2003) A teoria piagetiana da equilibração e as suas consequências educacionais. Disponível em: https://bibliotecadigital.ipb.pt/handle/10198/208

Fonseca, M., Campos, P., &Gonçalves, D. (2012) Introdução ao Design de Interfaces. FCA.

Google. (n.d.). Exploring computational thinking. Disponível em: http://www.google.com/edu/computational-thinking/

Gresse Von Wangenheim, C.; Nunes, V. R.; Dos Santos, G. D.; Alves, Nathalia da Cruz; Coan; Evandro Sperfeld; Mansur, Camile. (2014). *Resumo de objetivos de aprendizagem de computação no ensino fundamental (Currículo de Referência CSTA/ACM K-12)*. Disponível em: http://www.computacaonaescola.ufsc.br/wp-content/uploads/2013/09/CurriculoACMIEEE-resumido-PORT_v10.pdf

Grover. S. (2014). *Foundations for Advancing Computational Thinking: balanced designs for deeper learning in an online computer science course for middle school students*. Ph.D. Thesis, Stanford University.

Grover, S.; Pea, R. (2013). *Computational Thinking in K−12 : A Review of the State of the Field*. Educational Researcher, 42(1), 38–43.

Guttormsen, S. (2010). Short Overview and Summary of Learning Theories. Disponível em: http://studmed.unibe.ch

Guzdial, M., (2008). Education: Paving the way for computational thinking. *Commun. ACM*, 51(8): 25-27.

Guzdial, M. (2015). *Learner-Centered Design of Computing Education: Research on Computing for Everyone (Synthesis Lectures on Human-Centered Informatics)*. Morgan & Claypool Publishers.

Hourcade, J.P. (2015). Child-Computer Interaction. Iowa City, IA: Author.

Kafai, Y. B. (2006). Constructionism. In K. Sawyer (ed), *Cambridge Handbook of the Learning Sciences*. New York: Cambridge University Press, 35–46.

Kafai, Y. B., & Resnick, M. (Eds.). (1996). Constructionism in practice: Designing, Thinking, and Learning in a Digital World. Mahwah, NJ: Lawrence Erlbaum.

Koschmann, T. (1996). Paradigm shifts and instructional technology: An introduction. In T. Koschmann (Ed.), CSCL: Theory and practice of an emerging paradigm. (pp.1-23.) Mahwah, New Jersey: Lawrence Erlbaum Associates.

Lu, J.J. & Fletcher, G.H.L. (2009). Thinking about computational thinking. *ACM Special Interest Group on Computer Science Education Conference*, (SIGCSE 2009), (Chattanooga, TN, USA), ACM Press.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: A sneak preview. Proceedings of the Second International Conference on Creating, Connecting, and Collaborating through Computing, Kyoto, Japan, 104-109.

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. SIGCSE conference, Portland, March 2008.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transactions on Computing Education (TOCE), vol. 10, no. 4 (November 2010).

Marques, M.T.M. (2009) Recuperar o engenho a partir da necessidade, com recurso às tecnologias educativas: contributo do ambiente gráfico de programação Scratch em contexto formal de aprendizagem. Tese de Mestrado. Universidade de Lisboa.

Maxwell, John W. (2006). "Resituating Constructionism." In International Handbook on Virtual Learning Environments, Volume I. Ed. J. Weiss, J. Nolan, J. Hunsinger & P. Trifonas. Netherlands: Springer. 279–298.

Monroy-Hernández, A. (2012) Designing for remixing: Supporting an online community of amateur creators (Doctoral dissertation, Massachusetts Institute of Technology).

Microsoft (2007). Microsoft, Carnegie Mellon Establish Center for Computational Thinking. Disponível em: https://news.microsoft.com/2007/03/26/microsoft-carnegie-mellon-establish-center-for-computational-thinking/#sm.0001jta0vlie1f5xqjo2kg38k2zmc

Ministério da Educação (2007). Educação e Formação em Portugal. Disponível em: http://www.dgeec.mec.pt/np4/97/%7B$clientServletPath%7D/?newsId=147&fileName=educacao_formacao_portugal.pdf

National Research Council (2010). *Report of a Workshop on the Scope and Nature of Computational Thinking*. Disponível em: http://www.nap.edu/catalog/12840.html

National Research Council (2011). *Report of a Workshop of Pedagogical Aspects of Computational Thinking Committee for the Workshops on Computational Thinking*. Disponível em: http://www.nap.edu/catalog/13170

Ng, W. (2015). Theories Underpinning Learning with Digital Technologies. In New digital technology in education: Conceptualizing professional learning for educators (pp. 73-94).

Nielsen, J. (2012) Usability 101: Introduction to Usability. Disponível em:

http://www.nngroup.com/articles/usability-101-introduction-to-usability/

Norman, D. A. (2013). The Design of Everyday Things. Basic Books.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Papert, S., (1991). Situating constructionism. In S. Papert & I. Harel (Eds.),

Constructionism. New York: Ablex Publishing.

Papert, S. (1999). Papert on Piaget. "The century s greatest minds." Time March 29, p. 105.

Pearson, K. (2009). How Computational Thinking is changing journalism & what's next? Disponível em: http://www.poynter.org/2009/how-computational-thinking-is-changing-journalism-whats-next/95941/

Philips, P. (2008). *Computational Thinking: A Problem-Solving Tool for Every Classroom*. Disponível em: http://education.sdsc.edu/resources/CompThinking.pdf

Ramos, J. L., & Espadeiro, R. G. (2014). Os futuros professores e os professores do futuro. Os desafios da introdução ao pensamento computacional na escola, no currículo e na aprendizagem. *Educação, Formação & Tecnologias*, 7 (2), 4-25. Disponível em: http://eft.educom.pt.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools, In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE 2010, (Milwaukee, Wisconsin, USA, March 10 - 13, 2010), ACM, New York, NY, 265-269.

Peters, D. (2013). Interface Design for Learning: Design Strategies for Learning Experiences. New Riders (Voices that Matter).

Quintana, C., Krajcik, J., & Soloway, E. (2000). Exploring a Structured Definition for Learner-Centered Design. In B. Fishman & S. O'Connor-Divelbiss (Eds.), Fourth International Conference of the Learning Sciences (pp. 256-263). Mahwah, NJ: Erlbaum.

Quintana, C., Norris, C., Krajcik, J., & Soloway, E. (2003). A Framework for Understanding the Development of Educational Software. In: A. Sears & J.

Jacko (Eds.), Handbook of Human-Computer Interaction, Lawrence Erlbaum & Associates.

Quintana, C., Soloway, E., & Krajcik, J. (2003). "Issues and Approaches for Developing Learner-Centered Technology". In M. Zelkowitz (Ed.), Advances in Computers, Academic Press, (Vol. 57, pp. 271-321). New York: Academic Press.

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., & Eisenberg, M. (2005). Design Principles for Tools to Support Creative Thinking. National Science Foundation workshop on Creativity Support Tools. Washington DC.

Resnick, M., Silverman, B. (2005). Some Reflections on Designing Construction Kits for Kids. Proceedings of Interaction Design and Children conference, Boulder, CO.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y. (2009). Scratch: Programming for all. Communications of the ACM , 52 , 60–67.

Resnick, M. (2013). Learn to Code, Code to Learn. EdSurge, May 2013. Disponível em: https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn

Resnick, M., Rosenbaum, E. (2013). Designing for Tinkerability. In Honey, M., & Kanter, D. (eds.), Design, Make, Play: Growing the Next Generation of STEM Innovators, pp. 163-181.

Resnick, M. (2014). Give P's a Chance: Projects, Peers, Passion, Play. Constructionism and Creativity conference, opening keynote. Vienna.

Resnick, M., Siegel, D. (2015). A Different Approach to Coding. Bright/Medium. Disponível em: https://medium.com/bright/a-different-approach-to-coding-d679b06d83a#.p3vc9n663

Royal Academy of Engineering (2012). Shut down or restart? The way forward for computing in UK schools. Royal Society. Disponível em: https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf

Sáez-López, J.M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using “scratch” in five schools, Computers & Education, 97, 129-141.

Scratch 2.0: Cultivating Creativity and Collaboration in the Cloud. Proposal to the National Science Foundation, 2010 (project funded 2010-2013). Co-PI: John Maloney, Natalie Rusk.

Selander, S. (2016). Conceptualization of Multimodal and Distributed Designs for Learning. In: The Future of Ubiquitous Learning: learning Designs for Emerging Pedagogies / [ed] Begoña Gros, Kinshuk, Marcelo Maina, Berlin: Springer.

Selby, C.; Woollard, J. (2014). *Refining an understanding of computational thinking. Technology, Pedagogy and Education*. Sage. Disponível em: http://eprints.soton.ac.uk/365507/1/140606RUofCT_TPE_SelbyWoollard.pdf

Siemens, G. (2005) Connectivism: A Learning Theory for the Digital Age. Disponível em: http://www.itdl.org/journal/jan_05/article01.htm

Soloway, E., Guzdial, M., & Hay, K.H. (1994) “Learner-Centered Design: The Challenge for HCI in the 21st Century”. Interactions, Vol. 1, No. 2.

Sousa, R. M., Lencastre, J. A. (2014). Scratch : uma opção válida para desenvolver o pensamento computacional e a competência de resolução de problemas. Disponível em: http://repositorium.sdum.uminho.pt/handle/1822/29944

Wing, J. (2006). Computational thinking. *Communications of the Association for Computing Machinery*, 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, pp. 366(1881), 3717-3725.

Wing, J. M. (2011). Computational Thinking—What and Why? The Link - Carnegie Melon School of Computer Science. Disponível em: http://www.cs.cmu.edu/~CompThink/papers/TheLinkWing.pdf

Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. Journal of child psychology and psychiatry, 17(2), 89–100. Wiley Online Library.